Generic Architecture for Predictive Computational Modelling with Application to Financial Data Analysis: Integration of Semantic Approach and Machine Learning

Yerashenia, Natalia

https://doi.org/10.34737/w4037

PhD Thesis

Generic Architecture for Predictive Computational Modelling with Application to Financial Data Analysis: Integration of Semantic Approach and Machine Learning

Author
Natalia Yerashenia

Supervisors
Dr. Alexander Bolotov
Dr. Fang He

June 27, 2023

School of Computer Science and Engineering

UNIVERSITY OF WESTMINSTER

ABSTRACT

The PhD thesis introduces a *Generic Architecture* for *Predictive Computational Modelling* capable of automating analytical conclusions regarding quantitative data structured as a data frame. The model involves heterogeneous data mining based on a semantic approach, graph-based methods (ontology, knowledge graphs, graph databases) and advanced machine learning methods. The main focus of my research is data pre-processing aimed at a more efficient selection of input features to the computational model.

Since the model I propose is generic, it can be applied for data mining of all quantitative datasets (containing two-dimensional, size-mutable, heterogeneous tabular data); however, it is best suitable for highly interconnected data.

To adapt this generic model to a specific use case, an Ontology as the formal conceptual representation for the relevant domain knowledge is needed.

I have determined to use financial/market data for my use cases. In the course of practical experiments, the effectiveness of the PCM model application for the UK companies' financial risk analysis and the FTSE100 market index forecasting was evaluated. The tests confirmed that the PCM model has more accurate outcomes than stand-alone traditional machine learning methods.

By critically evaluating this architecture, I proved its validity and suggested directions for future research.

To my dearest grandpa, Anatoly Shevelev

Acknowledgements

Reflecting upon this doctoral journey, I realise it was not a solitary pursuit. It was enriched and made possible by a collective of exceptional individuals with unwavering support and encouragement.

First and foremost, I owe a debt of gratitude to my primary supervisor, Dr Alexander Bolotov. His unwavering faith in my abilities and his readiness to guide me at any time, often beyond conventional working hours, has been a beacon throughout this journey. His wisdom, patience, and moral support have been the driving force behind this work; I am forever grateful for this.

I am equally thankful for the insightful guidance provided by my secondary supervisor, Dr Fang He, during the early stages of my project. Her valuable inputs shaped the trajectory of my research.

Special thanks to David Chan You Fee, whose technical expertise and patience have been invaluable during our numerous online meetings. His dedication and generosity in sharing his coding skills during the COVID-19 lockdowns have been nothing short of exemplary.

My heartfelt appreciation extends to Dr Gabriele Pierantoni, who has been a professional colleague and a dear friend. His valuable advice and steadfast support throughout the project were invaluable. His camaraderie has made the journey all the more pleasant and enriching.

I express my warmest gratitude to all my colleagues at the School of Computer Science and Engineering. You've provided a nurturing environment fostering my professional growth and well-being. Your camaraderie and support have been invaluable.

Lastly, I owe an immeasurable debt of gratitude to my parents, Tatsiana and Aliaksandr. Their unwavering support, even in the face of my occasional moral breakdowns and seemingly crazy ideas, has been my strength. Their faith in me has always encouraged me to rise above the challenges. To my dear friends who have put up with my perpetual preoccupations and mourning, I offer my sincerest appreciation. You all have played an indispensable role in my journey.

This dissertation is not merely a testament to my academic endeavour but a tribute to all the wonderful people who have supported, inspired, and journeyed with me. Thank you.

Contents

1 Inroduction

1.1 Research Aims

In this research, a concept of an intelligent, analytical predictive system for quantitative data is proposed, designed to process information and carry out a comprehensive investigation of it during a particular time period.

The research aims to create a Generic Architecture for *Predictive Computational Model (PCM)* capable of automating analytical conclusions regarding the particular type of data, where information is presented reliably and objectively. This will significantly help to advance the data analysis strategy by increasing the level of consistency, reliability, and efficiency. A holistic approach will be applied, intended to achieve systematic, functional, technical, methodological and informational compatibility of the components of the analysis into a single whole.

The proposed model's main feature is consolidating information management with the decision-making process to serve the prediction purpose. This involves managing a vast amount of heterogeneous data based on semantic and graph-based methods (ontology, knowledge graphs, graph database) and advanced machine learning methods.

The main focus of my research is advanced data pre-processing aimed at a more efficient feature (in this sense, inputs) selection for the computation (machine learning) model's input.

To test the model, I need to feed it with real-life data, which requires expert knowledge in a particular area depending on the data type. It was considered to apply PCM for financial/market analysis. I have chosen this type of data due to the peculiarities of its characteristics – the ' big 4V + R' theory (see Section 2.3, and my published paper [1]).

Financial data, in this thesis, is the set of a company's financial indicators, reporting or calculated data characterising various aspects of the company's activities related to the formation and use of its monetary funds and savings and used for companies' financial analysis. Financial indicators are expressed in absolute and relative (ratios) values.

Market data includes information about stocks, debt, foreign exchange, commodities and money markets – both in real-time and fundamental information.

Financial & Market data embody *high interconnectivity* because of the intricate relationships between financial indicators, metrics, and variables. These connections are built on the foundations of business operations and the fiscal environment. For instance, a company's revenue influences its profitability, affecting its share price. Similarly, external factors such as interest rates or inflation can impact a firm's financial health, making financial data an intertwined web of information. In essence, changes in one aspect of a company's financials often cascade into various other areas, making the entire financial data system highly interconnected.

Financial & Market data also embody *heterogeneity*, which means it encompasses a wide range of different types of data. This diversity is seen in the vast array of financial metrics that can assess a company's financial health, such as ratios, trends, absolute numbers, and more. Additionally, financial data extends beyond internal company data, involving many external economic indicators like inflation rate, exchange rate, GDP growth, etc. The data sources can vary from structured financial statements to unstructured news reports or qualitative data like management commentary. This heterogeneity can also extend to the temporal dimension, with some variables changing daily (like stock prices, while others change only annually (like annual reports). All these factors contribute to the heterogeneous nature of financial & market data.

The thesis presents two *use cases*: the UK companies' bankruptcy level detection and FTSE100 index prediction.

In the context of this dissertation, *use case* is a detailed description of how a system or process should behave in response to a particular kind of input or user action. It is designed to illustrate functionality and behaviour under specific conditions. The use cases demonstrate how a particular theoretical approach or methodology can be applied in a practical context (specific scenario).

1.2 Research Objectives

To achieve the main aim of the research, the following major objectives were set.

1. To examine and compare the most popular existing data mining techniques, especially concerning the use case data – financial/ market data.

2. To identify how data pre-processing can contribute to data mining, especially for interconnected and heterogeneous data. To explore the application of the semantic methods in data pre-processing.

3. To investigate what components should be included in the computational model addressing the complex data mining problem. Additionally, to investigate how information should be transferred between the model components.

4. To develop a Semantic Database System (Semantic DS) as the core of PCM model. To investigate how to structure it best. It will be shown in further research that it considers three components: Ontology, Graph Database and Feature Selection components.

   (a) To develop Ontology, which defines the basic concepts of data analysis of a particular type and the objects that serve as sources of knowledge for it. To investigate how to apply the semantic approach to structuring data containing all the reference information used for the analysis. The reader will see that it will be created in two stages: an informal conceptual catalogue of all the terms (objects) and a logical-semantic functional model using the resource metadata description language, Resource Description Framework (RDF).

   (b) To develop Graph Database (Graph DB) linked to the Ontology; it is a tool for interconnected data storage and pre-processing. It utilises a powerful semantic data technology, which serves as a semantic data repository for the proposed model.

   (c) To identify the method allowing the selection of a set of key indicators that characterise the assigned task of the model most thoroughly and adequately. The Feature Selection component will select the most relevant indicators, as I will show further. It aims to perform a range of genuinely significant input indicators (features, ratios) from a broad array of information complicated by unnecessary (noisy) information. Further, these indicators will be served as essential variables for the model's input.

5. To develop a Machine Learning Engine (ML Engine) with semantic data input. It is proposed to use computational tools to analyse the input data: Classical Neural Network, LSTM, etc. Additionally, the ML Engine will allow estimating the level of importance of each input feature (features' weights). The adequacy and accuracy of each tool for particular model purposes should be assessed. The model developed will be tested on real-life data. Two use cases will be presented.

6. To develop a mechanism of data exchange between the structural parts of the Semantic DS and other system components, it is necessary to find a way to transfer data automatically.

7. To create a Feedback Loop for the model, which will allow transferring the information about the importance of each input feature from the ML Engine to the Semantic DS. It should increase the efficiency of feature selection and the final output in general.

8. Test the consistency and validate the entire model to ensure its reliability and accuracy in solving the research problem.

1.3 Original Contribution to Knowledge

The contributions of the research are the following.

In the earlier stages of my research, I explored various modern approaches to the automated processing of heterogeneous qualitative data related to data analysis & forecasting, which are primarily stand-alone machine learning techniques. As a result, it was revealed that the efficiency of the data mining process is directly related to how deeply the algorithm can analyse the input data. Thus, I developed, tested, and implemented my methodology by utilising and integrating a novel semantic data management approach, which can formalise human domain expertise, allowing better quality data storage, structuring and processing than classic stand-alone methods.

I introduced a novel *Generic Component-Based Architecture* for *Predictive Computational Model*, which integrates the Semantic Database System and comprehensive machine learning algorithms.

Semantic Database System is my original development, which comprises the ontology and the graph database.

The ontology allows more efficient processing of complex and diverse interconnected data; it embodies a special conceptual framework for data analysis based on human domain expertise, which can be read by users and computers (i.e., model components or external applications).

A model can use a third-party Ontology, or the user can develop their own Ontology (if they have enough experience in the field). For example, I created two unique Ontologies for the use cases – *Ontology of Bankruptcy Prediction* (see Section 4.2) and *Ontology of Market Index Prediction* (see Section 5.2).

The graph database based on ontology enables a more comfortable and more efficient way of storing and sharing structured data; however, its primary benefit is performing complex feature selection queries.

Thus, the data necessary for practical analysis is structured and processed automatically.

I use a machine learning approach, which significantly reduces the processing time, increases the accuracy of the result, and eliminates human errors.

For example, in developing the Machine Learning Engine (ML Engine) for market index prediction (see Use Case 2 – Section 5.4), I used multiple parameters input related to predicting time-series data: the market index itself and the macroeconomic indices, used as additional external impact factors. It was a challenging research task because the *Long-Short Memory (LSTM)* neural network, the primary and one of the most accurate methods of forecasting time-series market data, is intended to analyse one input parameter only (dealing with time-series vectors). Therefore, to solve this unusual task of multiple parameters input, a novel Hybrid ML Engine was developed, consisting of several parallel LSTMs, a Concatenation Unit, which transfers the LSTMs' output into a Linear Regression (LR) Unit, which produces the final result.

Semantic Database System serves to pre-process data for the input to the Machine Learning component, improving the quality of input data and, consequently, the reliability of resulting values.

Addressing the quality enhancement of the system outcome, the *Feedback Loop* mechanism is introduced. It collects the information about input features' importance from the Machine Learning Engine when the model completes its first cycle and sends it to Graph Database, where this information is used for further advanced feature selection purposes.

1.4 Publications Resulting from the Thesis

Parts of the research in this thesis have been published in the following IEEE papers:

✓ Natalia Yerashenia and Alexander Bolotov. *Computational modelling for bankruptcy prediction: Semantic data analysis integrating graph database and financial ontology.* In 2019 IEEE 21st Conference on Business Informatics (CBI), volume 1, pages 84–93. IEEE, 2019.

✓ Natalia Yerashenia, Alexander Bolotov, Gabriele Pierantoni, and David Chan. *Semantic data preprocessing for machine learning based bankruptcy prediction computational model.* In 2020 IEEE 22nd Conference on Business Informatics (CBI). IEEE, 2020.

✓ Natalia Yerashenia, Alexander Bolotov, and David Chen. *Developing a Predictive Computational Model using semantic data pre-processing with machine learning techniques and its application for stock market prediction purposes.* In 2022 IEEE 24nd Conference on Business Informatics (CBI). IEEE, 2022.

2 Literature Review

This chapter embarks on an extensive literature review of the foundational concepts and techniques that scaffold the preliminary research for the proposed Predictive Computational Model. The objective of this review is threefold: to situate the research within the broader academic discourse, to reveal gaps in the current state of knowledge, and to form a theoretical underpinning that will guide the development of the PCM model in the subsequent chapters.

The review initiates with Section 2.1, delving into the expansive topic of data pre-processing in data mining. This fundamental initial step sets the stage for further data analysis. Section 2.2 ventures into the realm of feature selection methods, the judicious application of which can notably augment the performance of machine learning models.

The discussion in Section 2.3 narrows further to highlight the semantic approach to data pre-processing, a vital consideration shaping the PCM's structure.

The review then transitions to a rigorous examination of Ontologies and Graph Databases in Sections 2.4 and 2.5, respectively. This exploration unravels how ontologies offer a standardised conceptual library for information exchange and how Graph Databases handle data exhibiting intricate interrelationships.

Recognising the research's ambition to construct a computational model tailored for financial data applications, Section 2.6 devotes itself to exploring computational methods in financial analysis and forecasting. This section provides an overview of statistical and machine learning methods and acquaints readers with current software applications for data mining.

Finally, Section 2.7 presents a discussion on Python as the chosen development environment for the Machine Learning Engine (ML Engine), emphasising its versatility and widespread recognition within the machine learning community.

By exploring the contribution of data pre-processing to the analysis of interconnected heterogeneous data, identifying potential components for inclusion in the PCM, and comparing existing data mining techniques, this literature review effectively lays the groundwork for the forthcoming research.

2.1 Data-Preprocessing in Data Mining

Data obtained through the collection process must meet specific quality criteria. Data quality is a comprehensive concept that assesses their suitability for solving a particular problem [2]. ISO 9001 identifies completeness, reliability, accuracy, consistency, availability, and timeliness as the primary quality criteria [3]. Consequently, an essential sub-step of financial data analysis is data pre-processing.

The raw data obtained from corporate storage often needs a clearer structure. The initial data is frequently distorted and unreliable, containing values that exceed acceptable ranges (noise), abnormal values (outliers), and missing values [4]. Contrary to popular belief, machine learning cannot operate autonomously and independently. Like any IT tool, machine learning requires well-defined source data and instructions to ensure proper functioning. It is only feasible to load some accumulated Big Data of different formats into the machine learning algorithm and obtain accurate results.

One critical stage preceding the application of machine learning methods is data pre-processing, which involves various types of transformations. Recognising the importance of data preparation, standards such as Knowledge Discovery in Databases, CRISP-DM, and SEMMA treat it as a separate phase in Data Mining.

The analysis of various sources [5–9] indicates that the data pre-processing process should encompass the following tasks:

1. *Feature Selection.* This involves selecting only the most relevant features (e.g., financial ratios) and excluding the rest, considering their significance for data analysis objectives, quality, and technical limitations (e.g., volume and type).

2. *Data cleaning.* This task includes removing typos, errors, incorrect values (e.g., numbers in string parameters), missing values, eliminating duplicates and different descriptions of the same object, and restoring uniqueness, integrity, and logical connections.

3. *Feature generation.* It involves creating derived features and converting them into vectors suitable for the Machine Learning model, as well as transforming the data to enhance the accuracy of machine learning algorithms.

4. *Integration.* This task involves merging data from various sources (information systems, tables, protocols, etc.), including aggregation, where new values are computed by summing information from existing records.

5. *Formatting.* This includes syntactic modifications that do not alter the data's meaning but are necessary for modelling tools, such as sorting in a specific order, removing unnecessary punctuation marks in text fields, trimming long words, rounding real numbers to integers, etc.

Hence, before applying machine learning algorithms, data must be converted into a tabular representation, which is common in Machine Learning and Data Mining [7]. Upon receiving raw data, such as in CSV format, the analyst examines it to understand the nature of records (rows) and attributes' meaning, type, and value range (columns). Subsequently, the data scientist creates a dataset and selects data potentially relevant to the machine learning hypothesis under investigation.

The next step involves cleaning the data using built-in data software tools. This stage aims to identify and eliminate inconsistencies to improve the dataset's quality [10]. Incorrect, duplicated, or missing information can lead to reliable statistics and accurate conclusions within a business context.

For numerical variables, *data normalisation* is applied to bring the datasets to a consistent range and enable their use in the same Machine Learning model [11]. Typically, data normalisation involves converting the original numerical values to a new range between 0 and 1 based on the initial minimum and maximum values.

Once independent predictors are identified and target features generated from them, the data scientist revisits the dataset to address multicollinearity issues, as they can increase the dimensionality of the Machine Learning model and cause overfitting [12]. Feature Selection methods are employed for this purpose.

Integration and formatting of the dataset are often performed using DBMS tools for data or other specialised tools designed for such operations (e.g., SAS [5]). In my case, I will employ graph databases for these tasks.

These actions, from sampling to data sorting, are repeated multiple times until the dataset becomes suitable for modelling, considering the characteristics of the selected machine learning algorithms and the hypothesis being tested.

Numerous studies have established the beneficial impact of data pre-processing on the accuracy of machine learning algorithms [11, 13–15]. Yet, it's important to note that earlier investigations into independent bankruptcy prediction models seldom integrate data pre-processing as a built-in model component.

2.2 Feature Selection Methods Overview

Even after cleaning, normalising, and transferring the data into a properly structured format, the input sample is not quite ready for machine learning analysis yet. For effective analysis, only those variables are needed that affect the final result [16].

The term *"curse of dimensionality"* usually refers to the difficulties associated with fitting models, estimating their parameters, or optimising a multivariate activation function in a large sample [17]. As the dimension of the input data space increases, it becomes more difficult to find global optima for this space. Therefore, there is a practical need to select from a large set of input variables those most valuable in predicting the outputs of dependent output variables [18].

Feature selection is a procedure for discarding insignificant variables from a cleaned sample before running machine learning and data mining [19]. Reducing the number of predictors is necessary for several reasons [19], [16], [18]:

- *Significance of features.* Usually, the initial sample always contains a lot of "noisy data": noise, outliers, and only some predictors affect the real result;
- *Solution accuracy.* Some machine learning algorithms are sensitive to the size of the input vector. For example, for neural networks, a large number of inputs can lead to overfitting;
- *Computation speed.* The fewer variables the activation functions contain, the faster the calculations will go.

Thus, reducing the dimension of the problem is a necessary step in data preparation, which is of decisive importance for the final result.

According to the number of studies, e.g., [20], [21], feature selection methods can be divided into several categories:

Filtering methods is a feature selection technique used in machine learning and data preprocessing. These methods aim to select a subset of relevant features for use in model construction [20]. Filter methods are called such because they "filter out" the less useful features from the dataset before the learning algorithm is run.

These methods evaluate features based on inherent characteristics independent of any machine learning algorithm. The characteristics generally evaluated include correlation, mutual information, statistical tests for dependency, and variance. The features that meet specific criteria are selected or filtered out based on their scores.

The most popular methods in this group are Information Gain (IG-indexing), Chi-Square Test, and Minimum Redundancy Maximum Relevance (mRMR), which allow ranking features by significance, assessing the degree of correlation between each of them and the target variable.

Information Gain (IG) calculates the entropy reduction resulting from transforming a dataset [22]. It can be used for feature selection by estimating the information gain of each variable in the context of the target variable.

Chi-Square Test is used for categorical features in the dataset. The chi-square is calculated between each feature and the target, and then the desired number of features with the best performance is selected [23].

Correlation coefficients (e.g., mRMR, Pearson's correlation, etc.) measure the linear relationship between two or more variables. With their help, we can predict one variable through another [24]. The logic behind using this method to select features is that the "good" variables are highly correlated with our target. It is worth noting that the variables should associate with the target but not correlate.

Filtering methods are reasonable because they are fast enough: they have low computational cost, which depends linearly on the total number of predictors. However, they consider each attribute in isolation, not considering their mutual influence on each other and the target variable in general. Therefore, the accuracy of modelling with these feature selection methods needs to be higher.

Wrapper methods are a class of feature selection techniques that involve selecting the optimal subset of features through a search algorithm guided by the performance of a specific machine learning algorithm [25]. The concept of "wrapping" comes from the idea that the feature selection process is "wrapped" around the ML algorithm, hence tightly coupling the feature selection with the model's predictive performance.

The main advantage of wrapper methods is that they can potentially find the optimal feature subset for a specific model, thereby often providing high model performance. However, the disadvantages of these methods are equally significant. First, they tend to be computationally expensive due to the need to train a model for every subset of features examined. Second, they are prone to overfitting because they closely tailor the feature set to a particular model [25].

Embedded methods integrate the feature selection process within the machine learning algorithm itself. They are called "embedded" because feature selection occurs inherently during the model training process. This class of feature selection techniques combines the strengths of both filter and wrapper methods, offering a balanced trade-off between computational efficiency and selection performance [20].

These methods generally provide a good trade-off between computational efficiency and selection performance [26]. They are less prone to overfitting than wrapper methods as they do not exhaustively search the feature space. Moreover, they are more computationally efficient since they combine the learning phase with feature selection.

Embedded methods, like decision trees and Lasso regression, have found extensive use in data mining tasks [19]. Decision tree-based methods inherently perform feature selection by evaluating node splits, making them useful for data mining tasks like customer churn prediction. On the other hand, Lasso regression imposes a penalty on the magnitude of the coefficients, encouraging a sparse solution and thus feature selection, which can be highly beneficial in high-dimensional data mining tasks.

LASSO regression adds a penalty to various model parameters to avoid overfitting [27]. A penalty is applied to the coefficients multiplying each predictor when regularising a linear model. Lasso regularisation has the property of reducing some coefficients to zero. Therefore, such features can be removed from the model.

Random Forest Importance. The tree-based strategies used by Random Forests naturally rank on how well they improve the cleanliness of the model in terms of data [28]. Thus, we can select the essential features by "pruning" the trees below a particular factor.

Incorporating embedded feature selection concepts will be pivotal in developing the PCM model. As an illustration, in the use cases, the PCM model integrates an in-built *Random Forest Feature Importance* component (see Sections 4.4 and 5.5).

2.3 Semantic Approach to Data Pre-processing

A Semantic Approach Definition. A semantic approach to data mining refers to interpreting and understanding data based on the intrinsic meaning, context, and relationships between the data elements than solely relying on raw data values or syntactic structures [29].

The semantic approach leverages metadata, taxonomies, ontologies, and other knowledge sources to understand the meaning of data, how different data items are related, and how they can be effectively combined for insightful analysis. It's used to enhance data quality, facilitate data integration from different sources, and improve the efficiency and accuracy of data analysis processes.

Here is a more detailed look at the critical aspects of the semantic approach.

One of the foundational aspects of this approach is integrating domain-specific knowledge into the data mining process. Incorporating such knowledge provides numerous advantages, including guiding the mining process, reducing the search space, validating the results, and offering a deeper understanding of the patterns discovered [30].

Another significant aspect of semantic data mining revolves around ontology-driven mining [31]. Ontologies offer a structured, formalised way to represent domain knowledge and prove instrumental in guiding the mining process, defining concept hierarchies, and interpreting the results. Furthermore, ontologies have shown efficacy in supporting multi-relational data mining, encompassing diverse entities and relationships.

The aim of semantic pattern discovery, a key component of semantic data mining, is to discern patterns that possess not only statistical significance but also semantic meaning. For instance, in semantic association rule mining, the objective transcends beyond the discovery of mere statistical co-occurrences between items, focusing instead on unearthing associations that hold meaningful semantic relationships [30].

Semantic data integration is integral to semantic data mining, addressing data integration from various sources and presenting it in a unified, semantically consistent format. This integration capability helps manage heterogeneous data, resolves data conflicts, and provides a holistic view of the data for mining [32].

Lastly, an emphasis is placed on achieving interpretable and actionable results. Semantic data mining strives to yield outcomes that are not only easily understandable but also actionable. Therefore, the patterns or models discovered should be meaningful, comprehensible, and applicable within the specific context of the application or decision-making process [30].

Semantic data mining, therefore, offers a more efficacious, efficient, and insightful approach to discovery processes, particularly when contending with complex and heterogeneous data sources. It allows for more effective data integration, efficient search and query capabilities, and insightful data analysis, including trend detection, anomaly detection, and predictive modelling.

Features of the Input Financial data. According to Tsay [33], financial data is inherently intertwined, comprising many variables that interact in intricate and often non-linear ways. The value of a stock, for instance, hinges not only on the specific company's financial health, reflected in parameters like sales, costs, and profits, but also on broader economic indicators, industry trends, and even investor sentiment. External influences, including geopolitical events, further convolute these relationships, adding layers of complexity to an already dense network of interdependencies.

Such extensive relationships underscore financial theories like Modern Portfolio Theory (MPT), which explores how risk-averse investors can optimise returns given market risk levels [34]. MPT underscores the importance of diversification, premised on imperfectly positive correlation coefficients between asset returns. Additionally, financial time series data exhibits "stylised facts," such as volatility clustering and the leverage effect, illustrating financial data's inherent complexity and a high level of relationships.

Ontologies potentially provide a systematic and formalised approach to depict the interconnected nature of financial data [35].

Financial data is often called heterogeneous because it can comprise many different types of data and information, showing a high degree of variation [36]:

- *Multiple data types.* Financial data includes numerical data (like stock prices, revenues, and profits), categorical data (like credit ratings), ordinal data (like risk levels), and text data (like news reports and social media posts).

- *Different time frames.* Some financial data is updated every second (like stock prices), some every quarter (like company earnings), and some only once a year (like GDP numbers).

- *Various data sources.* Financial data can be sourced from stock exchanges, financial institutions, government reports, and various Internet sources.

- *Diverse sectors and regions.* Financial data covers many different sectors (like technology, healthcare, and utilities) and regions (like North America, Europe, and Asia).

  Because of all this variation, analysing financial data requires sophisticated models that can handle its heterogeneous nature.

For example, the components of a company's financial system can be (and usually this is the most common practice) described in the form of relational tables (traditional database), e.g. it is easy to present a balance sheet or income statement in such a way. However, to show the interconnections between all elements of these tables, it is necessary to create several tables of different structures containing thousands of objects. In this case, database management and search efficiency are substantially affected. For example, it becomes problematic to formulate a general query for several databases because of the difference in objects and attributes of the domain or changes in objects over time. When the data are inserted, updated or deleted, the integrity constraints for the database with changing objects should be checked and assured that the data will be consistent after all modifications [31]. Also, as mentioned before, there is a problem with integrating new nodes into the system. When adding a new node, it is essential to check the data and the data schema for consistency with the information already available in the system [37].

These observations further support using graph databases, specific non-relational NoSQL repositories, for storing computational model data. Graph databases are well-suited for handling interconnectedness, as highlighted in prior research [38].

2.4 Ontologies as a part of Semantic Approach to Data Analysis

The task of automated exchange of models' formal descriptions was the primary motivating factor of ontological research.

Knowledge Base Concept. A *knowledge base* refers to a structured repository of information that stores, organises, and manages a collection of data, facts, rules, and relationships about a particular domain [39]. This domain can be wide-ranging, from a scientific discipline to an industry sector, and the knowledge base aims to support decision-making processes, problem-solving, or, more generally, to enable reasoning over its content.

In artificial intelligence (AI) and expert systems, a knowledge base plays a crucial role as it embodies the 'knowledge' these systems use to interpret, understand, and respond to queries or perform tasks within their domain of expertise.

Moreover, with the advent of ontology-driven approaches [40], knowledge bases have evolved to contain formal descriptions and semantic structures. An ontology provides a shared and common understanding of a domain that can be communicated across people and application systems. In this sense, an ontology forms the backbone of a knowledge base, offering a formal, explicit specification of a shared conceptualisation, including the objects, concepts, and other entities presumed to exist in some area of interest and the relationships among them.

Ontologies are often leveraged to create a universally comprehensible format for knowledge bases [41]. This format enables human understanding and automated understanding and interoperability between different software systems or knowledge bases. Hence, the formal ontological description of a knowledge base is essential for integrating and combining different expert systems, allowing them to communicate, share, and synchronise their knowledge effectively.

For example, if financial ratios are submitted to the financial analysis expert system, after comparing the facts of its knowledge base, it can declare a conclusion on the company's state. Also, the expert system can be further trained by adding new facts. Suppose two such expert systems are made by different manufacturers and in different companies. It was decided to create a new expert system based on two existing ones. If the experts were people, not machines, everything would be simple: they would discuss it. As for the software, the situation is different. Therefore, it is necessary to formally describe the contents of the knowledge base of each expert system so that this description is clear to the software that implements the combination of the two systems.

When developers aim to supplement a knowledge base with new facts continually, it becomes essential to systematise the content of that knowledge base. This involves segmenting the knowledge base, describing the contents of each segment, and defining ways to link these segments together. However, systematising knowledge is a challenging task. In addition to the knowledge directly contained in the database, there is also a need for "general knowledge" [42]. This general knowledge is often taken for granted by humans, but it is not accessible to a knowledge base containing only specific knowledge. Therefore, it is necessary to formally describe the basic properties of the system, known as top-level ontologies. In typical knowledge bases, there are at least three levels of knowledge: general or abstract knowledge described in a top-level ontology, knowledge of a specific subject domain, and specific knowledge added to the knowledge base by users or software agents.

Ontology Definition and Features. In computer science and information science, *ontologies* refer to a formal representation of knowledge as a set of concepts within a domain and the relationships among them. They are designed to model a domain of knowledge or a part of the world, embodying a view of what exists [43].

An ontology is structured around objects, concepts, entities, and relations. It provides a shared vocabulary, which can be used to model a domain – the type of objects and/or concepts that exist and their properties and relations. Essentially, ontologies capture both the semantics and structure of a domain.

According to Smiths & Ceusters [43], The primary purpose of an ontology is to enable communication and sharing of information in a precise, unambiguous way. They are mainly used in artificial intelligence, the semantic web, software engineering, biomedical informatics, library science, and many other fields where domain knowledge must be shared explicitly.

The term "ontology" migrated from philosophy to computer science in the 1980s, where it was used in various kinds of research on Artificial Intelligence (AI). Towards the late 1990s, it became actively used in information integration, internet search, and knowledge management. Later, ontologies were recognised as a crucial element in creating the Semantic Web, representing a new stage in developing the World Wide Web [31].

The *Semantic Web* is a vision of the World Wide Web where data is given explicit meaning, making it easier for machines to process and integrate information on the web automatically. The idea, according to Tim Berners-Lee (one of its primary architects) [44], is to create a web of data that can be processed directly and indirectly by machines.

In addition to hypertexts, Semantic Web contains semantic (meaning) descriptions of these documents and various services that present these documents to users. Understanding the contents of Web documents by computers enables programs to conduct a more accurate search of information on the Web, automatically organising this information and exchanging it with other applications.

The use of ontologies is a crucial component of the Semantic Web's infrastructure, and they form the basis for the creation of the Resource Description Framework (RDF) and the Web Ontology Language (OWL), which are fundamental standards in the Semantic Web stack (see the section below).

In summary, ontology is a formal expression of conceptual knowledge within a specific subject area [45]. Its significance can be compared to the knowledge base of an intelligent information system, and its construction resembles a specific form of human thinking. Human thinking in cognition operates with judgments, statements, concepts, and their relationships. This forms the basis for constructing an integral part of a theory—the ontological knowledge base within a given subject area [46]. Moreover, such knowledge is described in a declarative form.

The term "ontology" was first mentioned as a technical term in computer science by Gruber [47]. Gruber addressed the engineering problem of creating a mechanism for the interaction of knowledge-based software systems. To solve this problem, he proposed several methods:

- Highlighting the level of declarative knowledge in knowledge management systems. Declarative knowledge refers to a description of what the world consists of in a particular system, as opposed to procedural knowledge, which involves logical reasoning based on the system's description. Gruber suggested separating knowledge from its processing.

- Providing a knowledge base with a description of its declarative knowledge to other knowledge management programs. This information should be comprehensible to both humans and machines and thus can be issued in two forms:

    1. Canonical form represents the knowledge described in the language of predicate logic.
    2. Ontological form, which Gruber represented then as a set of term descriptions (classes, relations, constants, etc.) and definitions that connect these terms.

- Building a library of ontologies based on the selected descriptions that can be used in various knowledge bases.

Gruber emphasised that conceptualisation occurs through classes and attributes [47]. The nature of the problem being studied is represented by concepts described by classes, their properties (attributes), and specific objects that are instances of these classes.

The purpose of an ontology is to accurately describe conceptualisation by constraining the possible interpretations of non-logical symbols of a logical language, thereby establishing consensus on how to represent knowledge using that language [48]. Conceptualisation is seen as a set of informal rules that limit the structure of a part of reality.

Conceptualisation allows moving from primary theoretical concepts to increasingly abstract constructs, expanding the entire structure of the scientific approach and integrating it into broader disciplinary contexts.

Gruber also used the term conceptualisation to refer to the process of constructing an ontology [47]. The result of conceptualisation is called a conceptual scheme (framework). This term is used in databases and has nearly the same meaning but is related to determining the database schema for some production tasks.

Due to the variety of problems solved with the help of ontology, they can be differentiated according to many characteristics. In practice, the main classification criteria are the purpose, the specification level and the formality of the ontologies.

There are four main ontology types by the purpose [49]:

The top-level ontology contains descriptions of general concepts that are not related to specific subject areas; that is, they apply to any of them. Such concepts can be "time", "space", "event", "action", etc.

Domain ontology describes terminology in various subject areas.

Task ontology describes the specific processes typical for various subject areas. For example, "banking transaction", "diagnostics", etc.

Application ontology combines the ontology of tasks and the domain ontology to focus on the particular concepts referred to these two ontologies needed for a specific purpose.

The next criterion for ontology classification is the specification level. The specification level of the ontology is determined by the extent of concept description details introduced in the ontology [50], [51]. The more restrictions on use and the more relations with other concepts the description of a concept contain, the more detailed it is.

In Fig. 1 the spectrum of semantic description techniques with increasingly strong semantics is given. The following terms provided in the graph should be clarified.

Taxonomy. A set of concepts with the class-subclass relation defined between them fits the concepts into a hierarchy (a taxonomy) [52].

Thesaurus. The specification level increases if the description of concepts organised in the form of a hierarchy (taxonomy) is supplemented with additional relationships with other concepts [51]. The set of input relations depends on the problem being solved and the subject area.

As it can be seen in Fig. 1, ontology has the highest level of semantic relationships formality.

Various authors propose different classifications of ontologies according to their purpose, but they all agree on the criterion that an upper-level ontology is mandatory [53]. Using them, it is supposed to solve the problem of determining the similarity of different ontologies with each other, and, consequently, the problem of the systems' interaction using different application ontologies. Application ontologies should expand the

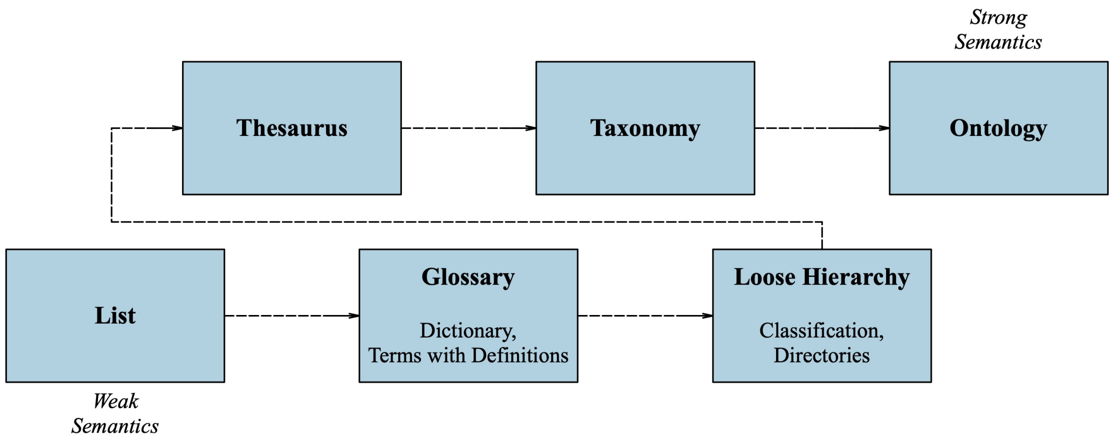


Figure 1: The process of Developing an Ontology

top-level ontology and refine it within a specific subject area. Ideally, there should be a single top-level ontology that would be used by all knowledge engineers to create an application ontology.

Given the extensive use of the term "declarative knowledge" in ontology theory, it is crucial to provide a detailed explanation. *Declarative knowledge* primarily comprises concepts, which serve as the fundamental elements of this knowledge framework [54]. Declarative knowledge representation revolves around describing concepts, the relationships among them, and their instances. These concepts form the foundation of both theoretical and practical knowledge, playing a vital role in capturing and organising knowledge within ontological frameworks.

Concepts denote sets (i.e. classes) of entities that belong to them [55]. Roles define the relationship between concepts. The term constructors are both Booleans and first-order operations such as quantifiers and operations that specify restrictions on roles, i.e., binary relationships. A primary logical concepts analysis framework in the logical representation of ontologies and reasoning about ontologies is *Description Logic* detailed in [56]).

Declarative knowledge is knowledge of facts, while procedural knowledge is how to apply these facts to solve problems [57]. Procedural knowledge is closer to actual activities but more challenging to describe and perform. Declarative knowledge is more natural to describe and less variable over time. It creates the framework (structure) of the knowledge space. Besides, declarative knowledge is fundamental since procedural knowledge is built on its basis. In this regard, declarative knowledge is mainly used to build a knowledge model.

Formal ontologies are described in a formal language with clearly defined syntax and semantics and possessing the properties of consistency and completeness. Logical languages are used as formal methods for writing ontologies, i.e. the logic of first-order predicates and its subsets [29] or description logic [56].

As already noted, ontologies are created primarily in order to determine knowledge in any subject area and exchange it between different users (organisations, software systems). Before the ontology is used it should be described in a language that a chosen system (application) understands.

Any ontology described in a language other than natural is partially or fully formal [29]. Thus, description logic enables use of ontology by software applications, since it limits the possible interpretations of syntactic constructions, excluding the ambiguity of the notation inherent in a natural language.

Ontology Presentation Format. The ontology presentation format is crucial in storing concepts and their relationships in the library. It transmits ontological descriptions to other consumers and provides mechanisms for processing its concepts. Specific ontology presentation languages have been developed to describe ontologies, with some of the most well-known ones being OWL, RDF, and KIF [58].

Resource Description Framework (RDF) provides a way to express statements in a format suitable for computer processing. RDF is a metadata description model, representing resources as a directed labelled graph. Each resource has properties, which can also be resources themselves or combinations of resources [59]. This allows RDF to describe both the structure of the resource and the associated subject area.

Initially, RDF was a language that allows describing information located on the Web; now it is a universal language for creating ontologies [60].

In the Semantic Web, these entities are called resources when talking about some Web entities. RDF is a language for describing such resources. Since computers should understand descriptions of the semantics of documents, it is necessary to develop special agent programs that would produce such a reading. Also, it is necessary to provide the ability to exchange information between various software agents. Thus, RDF means the language itself and various additional program modules necessary to ensure the full reading and exchange of information recorded in this language. This fact is emphasised in the name of the RDF language.

The main element of the RDF language is a triple [59]. A triple combines three entities: subject, object, and predicate. From a mathematical point of view, a triple is an instance of some binary relation. A relation over some domain D is a set of sequences of elements of D, and the number of elements in each sequence is fixed and equal to some predetermined natural number n. If $n = 2$, then the relation is called binary.

Since RDF is supposed to be used to describe resources distributed across different parts of the Web, it is necessary to solve somehow the problem of identifying the names of nodes and edges of an RDF graph, i.e. elements of triples. A standard approach is used for this: each element is described by the so-called *URI (Uniform Resource Identifier)* [60]. Usually, a URI is either a URL (Uniform Resource Locator) containing information about the location of a given resource on the Web, or a *URN (Uniform Resource Name)* that allows identifying a given resource in a certain namespace. A namespace is simply a named set of names and is used to make these names unique on the Web. For example, if there is a task to combine two RDF graphs (sets of triples), and at the same time, each of the graphs uses its node with the same local name, then I can solve the problem of identifying these nodes in the combination of graphs in two ways: give each of the nodes a unique name, using the URL of its RDF graph on the Web or to set its own namespace for each graph so the names of nodes with a local name will be unique in the union of graphs (since they will lie in different namespaces). In addition, URIs allow referencing the same Web entities from various RDF documents.

Web Ontology Language (OWL) is a language designed to describe ontologies and was developed by the W3 consortium specifically for this purpose. OWL is built as an extension of RDF and RDFS [61]. The basic syntax of the language is still XML, and the principal construct is the triple of the RDF language. In this context, OWL can be considered as an extended version of RDFS, which allows not only to describe classes and properties but also to set restrictions on their use. In the language of descriptive logic, this means that the logic based on OWL contains, axioms that specify the relations between these relations and various restrictions on the latter, in addition to the description of relations.

The process of creating ontologies relating to financial data was offered in the papers by Castells et al. [32], Grant & Soto [62] and Tang & Song [50].

Protégé as Ontology Development Environment Protégé is probably the most popular and oldest ontology editor at the moment. It is available for free download from the program website and can be supplemented with numerous plug-ins.

Ontologies in Protégé are created as so-called projects. A project is configuration data, first of all, the name of the OWL file in which the edited ontology is stored [63]. The configuration data also stores various editing settings associated with this project.

Protege allows not only to create domain ontologies but also to find errors in them [63]. Protege can be used as a rapid prototyping framework in which ontology developers can check semantic constraints and generate user interfaces.

In addition to standard ontology development tools, Protege has an open architecture that allows programmers to embed functional components using different plug-ins.

Some official plug-ins for Protege are included by default. For example, *OntoGraf* visualises the ontology as a graph tree, displaying only the approved axioms. *DL Query* and *SPARQ Query* plug-ins allow searching, analysing, reasoning, and filtering data through queries.

The software also allows us to logically conclude the editable ontology and set SPARQL queries to the ontology. To select the Reasoning main menu item, the following operations can be performed:

- To check the logical compatibility of the ontology (Check consistency).
- To classify individuals of ontology, ie build a taxonomy (Classify taxonomy).
- To compute new classes using declared class constructors (Compute inferred types).
- To set an ontology query in SPARQL.
- To change the inference programs with which the operations described above are performed.

The main advantage of Protege is in its simplicity of the controls for ontology description and management, as well as the ability to save the developed outcome not only as standard formats such as OWL and RDF, but such formats like XML schemas or JSON. Later, these files can be used as the framework for the computational models and managed using other environments, e.g. Python or Neo4j.

It should be noted that in comparison with relational databases, the ontology repository based on RDF triples suffer from poor performance. As mentioned above, a knowledge base based on RDF triples is a graph whose nodes are subjects and objects of RDF predicates, and edges are RDF predicates themselves. A quick search and complex queries on this graph are difficult to implement. However, the RDF file is an excellent base for creating a separate graph database.

2.5 Graph Databases Overview

Just one visual image can replace a thousand words. But a graph is much more than only an image. Graph theory provides a simple and powerful tool for constructing models and solving a wide range of classification and other data analysis problems [64]. This includes industrial production planning, network planning and control tasks, tactical and logical tasks, problems in building communication systems and researching information transfer processes, choosing optimal routes and flows in networks and methods for constructing electrical networks. Furthermore, graphs provide a solution to a wide range of business problems. Thus, the range of possible applications of graph theory is very wide.

The founder of graph theory is considered to be Leonard Euler, who in 1736 solved the problem of Konigsberg bridges [65].

Graphs are structures used to study paired relationships between objects, consisting of nodes and edges [66]. Nodes in graph databases correspond to database objects, and edges correspond to relationships between these objects. Both nodes and edges can have properties. Graph structure provides the best way to work with abstract concepts such as relationships and interactions. It also offers an intuitive visual way of thinking about these concepts.

Graph Databases, such as Neo4j, are examples of NoSQL databases designed explicitly for representing semantic data. They store, process, and visually represent standard structural elements. A typical graph database contains reference information about objects, eliminating the need for users/designers to search for this information in database directories. This reduces the potential for human errors and saves time in the design process [37]. Graph databases enable the automatic creation of standard elements, further reducing design time [38].

A graph database typically consists of two main components: a set of parameterised programs that create the necessary images (usually in dialogue mode) and a set of batch files that store all reference and auxiliary information related to the drawn elements [37].

NoSQL databases, including graph databases, are more efficient than traditional relational databases when quickly transmitting data over the network [67]. However, this efficiency is achieved by aggregating and working with data of specific types, which may result in disconnected aggregates. Overcoming this problem without using graph databases can be costly [68], whereas graph databases handle connectedness and relationships, which are essential features for financial data.

In financial analysis, identifying relationships between indicators is particularly important, and a graph database has been chosen as a tool for financial data pre-processing [68]. Unlike relational databases, where considering relationships can significantly reduce query performance for large datasets, graph databases maintain consistent performance regardless of the amount or variability of the stored data. One of the main reasons for choosing a graph database is its ability to process interconnected data efficiently, as queries in graph databases are localised in specific parts of the graph. The execution time of each query depends on the size of the graph section that needs to be traversed to fulfil the given request than the total size of the graph [38].

Another valuable feature of graph databases is their scalability. Adding new types of interconnections, nodes, labels, and subgraphs to the existing structure is easy without disrupting existing queries and functionality [38]. The data structure can adapt to the changing financial analysis needs and does not need to be predefined or remain unchanged. With the graph model's flexibility, it is unnecessary to predefine all potential scenarios in detail; the model can be adjusted in real-time for specific companies when using the Predictive Computational Model.

Lastly, graph databases provide the necessary tools for development and system maintenance [68]. They offer built-in graphical data models, along with user-friendly software interfaces (APIs) and query languages with clear semantics for each query, enabling effective application development.

Some of the most common graph database applications include *Neo4j*, *DEX*, *Titan*, and *OrientDB*, which are similar in usage. However, an empirical evaluation of these applications using a graph database benchmark tool for different workloads revealed that Neo4j has several advantages [69]. Additionally, an experimental comparison of these applications using the BlueBench Architecture, which assesses systems based on operations such as CreateIndexes, LoadGraphML, Traversal, ShortestPath, etc., showed that Neo4j, followed by DEX and Titan, outperformed the other systems due to the specialisation of their backends for this type of queries [70].

Neo4 as the Graph Database Development Environment The advantages of using graph databases instead of traditional ones have been outlined above. For storing, structuring, visualising and selecting data in my model, I use the Neo4j environment.

Neo4j[1] is an open-source graph database management system implemented in Java. Its developer is Neo Technology. This graph database environment stores data in a proprietary format specifically adapted for the presentation of graph information; this approach, in comparison with the modelling of a graph database, using a relational DBMS, allows for additional optimisation in the case of data with a more complex structure.

The initial development of this environment started in 2003, but it has been publicly available since 2007. The source code, written in Java and Scala, is available for free on GitHub or an easy-to-download application.

Neo4j uses its own query language, Cypher[2], though the queries can be done in other ways, for example, directly through the Java API. Cypher is not only a query language but also a data manipulation language, as it provides CRUD functions for graph storage.

In addition, Neo4j has both Community Edition and Enterprise Edition databases. The Enterprise Edition includes everything the Community Edition offers, plus additional corporate requirements such as backup, clustering, and failover.

Directed property graph is chosen as the data model in the Neo4j environment: Each graph database

[1] `https://neo4j.com/developer/graph-database/#_what_is_neo4j`
[2] `https://neo4j.com/developer/cypher/`

contains nodes and edges (links, relationships).

Nodes have properties. They can be thought of as documents containing properties in the form of key-value pairs. One or more labels can designate nodes. Labels group nodes by indicating the role they play in the dataset. Multiple labels can be assigned to a single node since nodes can play several different roles in different domains.

Edges connect nodes and structure the graph. They can also be named and directed (start and end nodes should be assigned). Besides, the edges can contain properties. This allows introducing additional metadata into the graph algorithms, adding various semantics to edges, and restricting queries in real-time.

The notable features of Neo4j that make this environment the most popular in its segment are listed below.

- In Neo4j, there is no need to follow a fixed pattern. Instead, we can add or remove properties as required. It also provides schema constraints.
- Neo4j fully supports ACID, a standard set of properties that guarantee the reliability of a database: atomicity, consistency, isolation, and durability.
- We can scale the database by increasing read / write operations and capacity without affecting query speed or data integrity. Neo4j also provides replication support for data security and reliability.
- Neo4j provides a powerful declarative query language known as Cypher. It uses ASCII art to render graphs. Cypher is easy to learn and can create and extract relationships between data without using complex queries.
- Neo4j provides a built-in Neo4j Browser web app. Using this, we can create and query graph data.
- Neo4j can interact with external environments and programming languages like Java, Python, etc.
- It supports two kinds of Java APIs: Cypher API and Native Java API for developing Java applications. In addition to this, we can also work with other databases like MongoDB, Cassandra, etc.
- Neo4j supports indexing with Apache Lucence.

The Neo4j developers' team is actively working on constant improvements to the application, often releasing updates. There is an active community of developers and professionals who share instructions and valuable advice on accomplishing specific tasks on the official website[3]. In addition, the website contains lots of helpful material and regularly published articles to encourage users. Finally, it includes examples of how actual companies use Neo4j[4]: social media, recommendation systems, financial fraud detection, and mapping systems.

[3]`https://neo4j.com/developer/graph-data-science/?role=datascientist`
[4]`https://neo4j.com/case-studies/`

2.6 Computational Methods in Financial Analysis and Forecasting

As it was mentioned before, the choice of machine learning method depends on the input data type. This Section will consider different statistical and machine-learning methods for assessing financial/market data.

When considering various computational methods of financial data analysis, it is proposed to use a two-level classification, which is presented in Fig. 2.

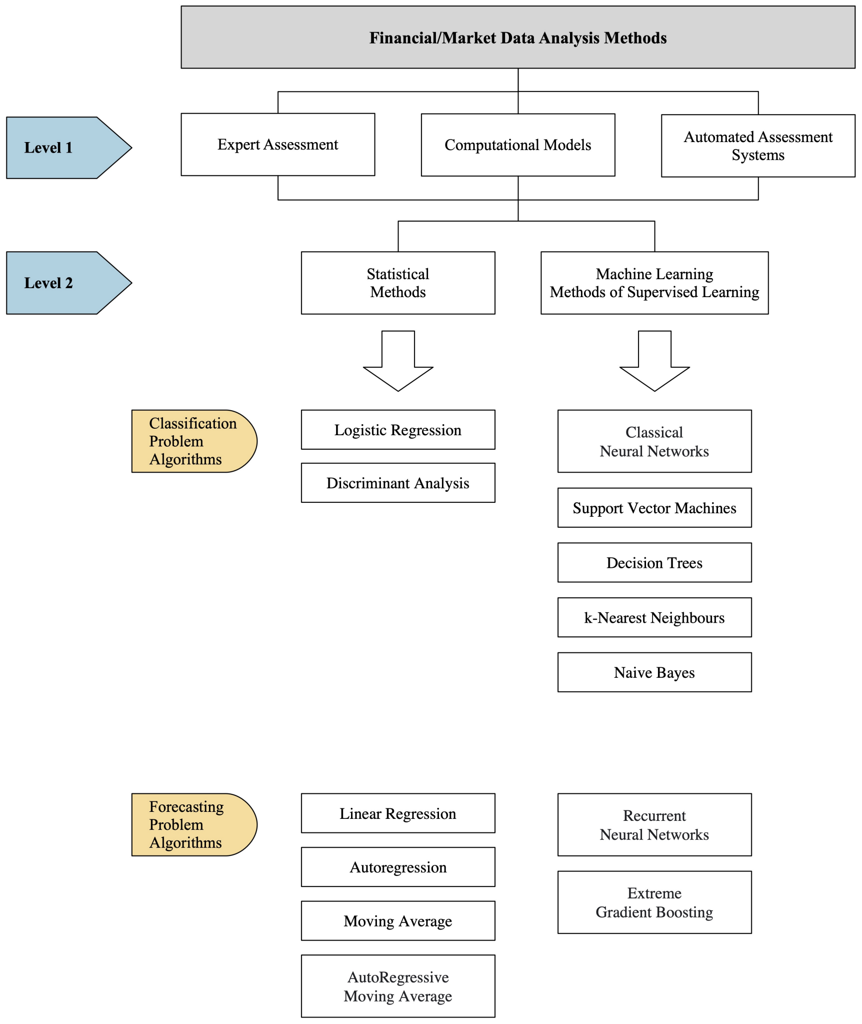

Financial/Market Data Analysis Methods
Level 1
Expert Assessment
Computational Models
Automated Assessment Systems
Level 2
Statistical Methods
Machine Learning Methods of Supervised Learning
Classification Problem Algorithms
Logistic Regression
Discriminant Analysis
Classical Neural Networks
Support Vector Machines
Decision Trees
k-Nearest Neighbours
Naive Bayes
Forecasting Problem Algorithms
Linear Regression
Autoregression
Moving Average
AutoRegressive Moving Average
Recurrent Neural Networks
Extreme Gradient Boosting

Figure 2: Classification of Financial/Market Data Analysis Methods

Methods of the first level include expert assessments, which are most often based on subjective opinion, and methods of automated assessment systems, i.e. analytical software.

Methods of the second level are mathematical-based support tools that include statistical and machine learning methods [71].

As mentioned before, the financial/market data mining process involves two basic types of analytical problems: classification and prediction. The methods used for these two categories are different.

For example, in Use Case 1 (see Section 4), the Companies' Bankruptcy Prediction relates to a classification problem. According to a particular combination of the company's financial ratios values, the system should define which type this company belongs to – bankrupt or non-bankrupt. In Use Case 2 (see Section 5), the aim is to forecast the future values of the market index price, so the analytical tool should resolve a prediction problem.

2.6.1 Statistical Methods in Financial Analysis and Forecasting

The idea of financial data analysis using a model-based approach became popular as early as the 1960s. Though, statistical methods were used as the basis.

Discriminant Analysis. *A discriminant analysis method* is based on the formation of training samples containing objects of various classes of object states (for example, normal, pre-crisis, crisis) [2]. Classification of the current state of a company can be carried out according to the values of financial criteria using specific rules - classification functions. The separation of the training sample objects belonging to a particular bankruptcy class from the objects of the other classes is carried out based on the dichotomy principle [72]. The value of the classifying function contains information about the class. It denotes the proximity to the border between objects of different classes, i.e., evaluates the risk degree. A probabilistic approach is used to determine the separating surfaces corresponding to the boundary values of the classification functions. It aims to construct models for recognising new objects.

In 1966, Beaver published a paper in which he compared the financial ratios of bankrupt firms with the performance indicators of companies that remained competitive [73]. He analysed the five years and 20 coefficients for a group of companies, half of which went bankrupt. Beaver's study showed that there were quite significant differences in the financial ratios of the two groups of firms. Bankrupt firms had a lower return on assets and return on sales, a higher proportion of accounts receivable, and lower values of current and absolute liquidity ratios but a higher level of debt.

A significant shortcoming of Beaver's method is the lack of consideration for the specifics of enterprises (for example, capital structure in various industries) and the economic situation in the country. At the same time, Beaver's prediction method was created based on data from the financial statements of American companies received back in the 1960s. Therefore, to use it effectively, it is required to collect and analyse new statistical data of the country and region in which it will be applied.

Altman, in 1968 took the next step in bankruptcy prediction [74]. According to Altman's approach concerning a specific country (USA) and to the time interval (the 1960s), a set of separate five financial ratios of companies was formed. The choice of these ratios was based on preliminary expert analysis. In the 5-dimensional space created by the selected coefficients, a hyperplane is drawn, which best separates successful companies from bankrupt companies, based on historical financial statistics. As a result, the well-known *Z-score model* appeared.

Altman's approach, also called the method of discriminant analysis, was subsequently applied by Altman himself and his followers in several countries (England, France, Brazil, etc.). For example, Taffler developed a model for UK companies [75].

The study of J. Ohlson used a significant number of companies (2163) to create a more accurate model [76]. Also, Ohlson was one of the first to use the logistic regression method instead of the discriminant analysis method. He proved that this method is more stable and reliable in the absence of the normal distribution of variables.

Regression Analysis. *Regression analysis method* is one of the most common methods of mathematical statistics in financial analysis, designed to determine the analytical interpretation of the stochastic dependence between the studied attributes [2].

Linear Regression (LR) is designed to *predict* continuous numeric variables. This method was invented back in 1800 by F. Galton [77], who investigated the relationship between the height of fathers and the height of their sons. Now, it's mostly used in prediction of market data, e.g. it is well-described in this article [78].

Regression is the conditional expectation of a continuous dependent (output) variable for the observed values of the independent (input) variables [79]. This method is based on the hypothesis that the sought dependence is linear. Thus, each independent variable contributes additively to the resulting value with some weight, called the regression coefficient [77].

Regression is called *simple* if there is only one input variable. However, such a model is too rough an approximation of reality, and in practice, as a rule, dependences on several variables are used – *multiple regression.*

Based on the analysis of several sources ([79], [80], [78], etc.), the advantages of the LR method include the following:

- The high speed and simplicity of obtaining the model.
- Interpretability of the model. The linear model is transparent and understandable for the analyst. Based on the obtained regression coefficients, one can judge how a particular factor affects the result and draw additional practical conclusions.
- Wide applicability. Many actual processes in economics and business can be described with sufficient accuracy by linear models.
- This approach is thoroughly studied. For LR, typical problems (for example, multicollinearity) and their solutions are well-researched, tests for assessing the static significance of the resulting models are developed and implemented.

If the output variable is *categorical or binary*, different modifications of the regression should be used. One such modification is *logistic (logit) regression*, designed to estimate the probability that the dependent variable will take a value between 0 and 1 [81]. In LR, the result (dependent variable) is continuous. Hence, it can have any of an infinite number of possible values. However, in logistic regression, the outcome (dependent variable) has only a few possible values.

LR uses *ordinary least squares method* to minimise errors and achieve the best possible fit, while logistic regression uses *the method of maximum likelihood* to obtain a solution [81].

Multivariate logistic regression analysis of the data allows considering the influence of nominal features by coding gradations of predictors by dichotomous variables that take a value of 1 for respondents belonging to the corresponding gradation and a value of 0 for the rest [82].

The task of assessing a company's financial position, which assumes that the response variable is binary (takes two values: crisis and non-crisis enterprises) can be effectively resolved with the help of logistic regression [83], [84].

It should be noted that none of the above statistical models for diagnosing bankruptcy is perfect since all of them are based on data from the past and are based on the specifics of the prevailing financial conditions. Besides, they reflect the features that formed the training sample. For example, when developing a Z-score, Altman studied companies whose shares were presented on the US stock market only. Moreover, out of 22 initially selected factor variables that could be useful for predicting bankruptcy, he ultimately left only 5, which were included in the model [74]. Therefore, the variables of each model take into account the industry specifics of functioning specific firms in a particular market at a particular time.

Thus, in light of the apparent inadequacy of the existing scientific methods for managing financial assets, the researchers are committed to using new approaches for predicting bankruptcy. Recently, methods of machine learning have supplanted traditional statistical methods.

More sophisticated methods for predicting financial/market time series are Autoregressions, a Moving Average method, Autoregressive Integrated Moving Average and

Before considering them, I should define the features of ***Time Series Analysis***. According to [33], there are two main goals of time series analysis: determining the nature of the series and forecasting (predicting future values of a time series from present and past values). Both of these goals require that the series model be identified and, more or less, formally described. Once the model is defined, it can be used to interpret the data (for example, use it to understand seasonal changes in stock prices). Finally, ignoring the depth of understanding and the theory's validity, it can be extrapolated the series based on the found model, i.e., predicting its future values.

Most of the standard components of a time series belong to two classes – *trend or seasonal components*. A trend is a common systematic linear or non-linear component that can change over time. The seasonal component is a recurring component. Both of these types of regular components are often present in a number at the same time.

The *Autoregression* (AR) predicts the next step in a sequence as a linear function of observations at previous time steps. The method is also suitable for one-dimensional time series without trend and seasonal components.

The autoregression process will be stationary only if its parameters are within a specific range. For example, if there is only one parameter, it must be in the range from -1 to 1. Otherwise, the previous values will accumulate, and the values of the subsequent ones can be unlimited; therefore, the series will not be stationary. If there are multiple autoregressive parameters, then similar conditions can be defined to ensure stationarity [85], [86].

Unlike the autoregressive process, in the moving average process, each element of the series is subject to the cumulative effect of previous errors.

The *Moving Average (MA)* method is one of the well-known methods for smoothing time series. Applying this method, it is possible to eliminate random fluctuations and obtain values corresponding to the influence of the main factors.

Moving average smoothing is based on the fact that random deviations mutually cancel out in averages [87]. This is due to the replacement of the initial levels of the time series with the arithmetic mean within the selected time interval [88]. The resulting value refers to the middle of the chosen time interval (period). Then the period is shifted by one observation, and the calculation of the average is repeated. In this case, the periods for determining the average are taken the same all the time. Thus, in each case under consideration, the mean is centred, i.e., the midpoint of the smoothing interval, and represents the level for this point. When smoothing a time series with moving averages, all levels of the series are involved in the calculations [89]. The wider the smoothing interval, the smoother the trend is. The smoothed series is shorter than the initial one by *(n - 1)* observations, where n is the size of the smoothing interval.

The *Autoregressive Moving Average (ARMA)* model proposed by Box and Jenkins [85] in 1976 includes both autoregressive and moving average parameters. The method models the next step in the sequence as a linear function of observations and random errors at previous time steps. It combines Autoregressive (AR) and Moving Average (MA) models.

The *Autoregressive Integrated Moving Average (ARIMA)* mode is a powerful tool for modelling and forecasting time series data. It combines autoregressive (AR), differencing (I), and moving average (MA) components to capture the temporal dependencies and patterns within the data. ARIMA is primarily used for univariate time series analysis, considering only the historical values of the target variable [85].

The Box-Jenkins approach to time series analysis is a powerful tool for accurate short-term forecasting. ARIMA models are flexible enough to capture a wide range of time series characteristics encountered in practice [90]. The formal procedure for checking the adequacy of the model is simple and accessible. Moreover, the fitted model can directly obtain forecasts and prediction intervals.

The *Autoregressive Integrated Moving Average with Exogenous Variables (ARIMAX)* expands upon the ARIMA framework by including exogenous variables, allowing for incorporating external factors that influence the target variable [91]. These exogenous variables can enhance forecasting accuracy by capturing the impact of relevant features on the time series of interest [92]. ARIMAX is particularly useful in financial forecasting, where economic indicators, interest rates, or market sentiment can significantly influence asset prices or market trends.

However, both ARIMA and ARIMAX models have certain limitations. One limitation is that these models assume linear relationships between variables and may not capture complex nonlinear patterns in financial data. Financial time series often exhibit nonlinear behaviours, such as sudden changes, volatility clustering, and non-constant variance, which may not be fully captured by the linear framework of ARIMA and ARIMAX [92].

Another limitation is the assumption of stationarity in the data. ARIMA and ARIMAX models are more suitable for stationary time series data, where the statistical properties remain constant over time. However, financial data often exhibit non-stationarity, such as trends, seasonality, or structural breaks, which require preprocessing techniques like differencing or transformations to achieve stationarity [93]. In such cases, the effectiveness of ARIMA and ARIMAX models may be limited, as these models may need help to capture the dynamics of non-stationary data adequately.

Furthermore, ARIMA and ARIMAX models are typically univariate, focusing solely on the target variable and its historical values. While including exogenous variables in ARIMAX enhances the model's capabilities, it may still need to fully capture the complexity of multivariate relationships in financial data [90]. Many interrelated factors influence financial markets, and considering only a limited set of exogenous variables may overlook crucial information or relationships that affect the target variable.

2.6.2 Machine Learning Methods in Financial Analysis and Forecasting

Classical Neural Networks. A neural network is an example of a machine learning method applied in bankruptcy prediction based on creating an artificially intelligent algorithm similar to the human brain. The first attempt to model these processes was the research of McCulloch and Pitts in 1943 [94]. The neuron of McCulloch and Pitts is a mathematical model of the biological neuron of the brain, taking into account its structure and functional properties.

It is worth mentioning Hebb's work [95], which provided one of the first models of training neural networks, as well as Rosenblatt's book [96], where a perceptron, the basic architecture of many neural networks, was proposed.

Neural networks have a significant advantage over traditional statistical data analysis algorithms. They are not programmed but trained. In the process of learning, the neural network detects dependencies between the input data and the results [97].

Numerous authors state the primary principles of neural network operation, such as [98], [99], and [100], etc.

Feed-forward (perception) neural networks – the most basic type of neural network for financial analysis – are formed of static neurons so that the signal at the output of the network appears at the same time that the signals are being inputted [101], [102]. A topology of such a network can be different [103]. For, instance, if not all of the neurons of the network are outputted, it indicates that the network contains hidden neurons. The most general type of network architecture is obtained when all neurons are connected to each other. In particular tasks, neurons are usually grouped into layers.

It is assumed that the information inflows *(X1, X2, ..., Xn)* pass through all layers sequentially, one after another, and are converted to the values of the output layer *(Y1, Y2, ..., Yn)*. The output is determined by the type of activation function. As it is stated in [101], neural networks with sigmoid functions are a universal tool for approximating functions.

It should be noted that an input layer and an output layer are prerequisites for constructing any artificial neural network; however, the number of hidden layers can be adjusted depending on the complexity of the processed data array [103].

Neural networks determine their benefits, firstly, from the ability to process information and, secondly, from the ability to learn, i.e. to create generalisations. The term generalisation refers to obtaining a reasonable result based on data that was not encountered in the learning process [104].

Self-organisation is one of the main distinguishing features of artificial neural networks. At the same time, the phenomenon of self-organisation is not unique and can be found in another important class of analytical constructions – intellectual multi-agent systems [105]. Multi-agent systems are also actively used in financial modelling and are often combined with artificial neural networks within the framework of a single model, e.g., [106], [107], [108] and [109], etc. In practice, being an autonomous technique, neural networks cannot provide ready-made solutions. To succeed in data mining, they must be integrated into complex systems. If the complex problem is divided into a sequence of relatively simple ones, then some of which can be solved by neural networks.

To minimise the error of the perceptron and obtain the desired output, an iterative gradient algorithm of backpropagation, should be used. The main idea of *backpropagation method* is to propagate error signals from the network outputs to its inputs in the direction opposite to the direct propagation of signals in normal operation [110]. Thus, the errors will be taken into account during the next iteration. When applying the backpropagation method, the derivative of the activation function needs to be found.

Artificial neural networks can be divided into two types: supervised learning and unsupervised learning. The first type means that the neural network should be provided with a certain set of training examples. Hence the potential output values of the model can be programmed, which should be as close as possible to the reference ones. Thus, initially, a neural network is taught using the training dataset (in the sample). Then, based on the optimised values, a decision is made on another, real data array (out-of-sample). Supervised learning neural networks include forward and reverse networks, recurrent networks, etc.

The analysis of numerous sources on the topic (e.g. [111], [100], [97], [102], [112], etc.) revealed the following main features, of using neural networks, which can be employed in bankruptcy prediction:

- *Nonlinearity.* Unlike conventional econometric models, neural networks process incoming signals non-linearly. Moreover, such a non-linear response is inherent in each element of the distributed system.

- *Adaptability.* Neural networks can adapt their synaptic weights to environmental changes. In particular, neural networks trained to operate in a specific environment can be easily retrained to work in conditions of slight fluctuations in environmental parameters. Besides, working in a non-stationary environment, neural networks have the ability to modify synaptic weights in real-time.

- *Iterativeness.* The information presented by a neural network, as a rule, is processed inside its structure by numerous times and in various ways (depending on the chosen training method). This significantly increases the likelihood of network convergence and the attainment of the desired optimum (for example, minimising errors). Plus, it becomes possible to fine-tune the network to a specific data array.

- *Context information.* All its other neurons can potentially influence each neuron of the network. This feature is not inherent in all neural networks, but only in advanced models (for example, Haken networks, recurrent networks). The use of contextual information is an important advantage of artificial neural networks in information processing, bringing it closer to biological neurons. The reaction of such a network is caused not only by specific input values, but also by their order, so the network can effectively process information even from repeating time series. In addition, the output value of such a network depends on its predictions in the previous steps.

- *Scalability.* The parallel structure of neural networks potentially accelerates solving some problems.

After the development of learning algorithms, the resulting models were used for various practical purposes: pattern recognition, control problems, and forecasting problems. The ability to simulate non-linear processes, work with noisy data and adaptability make it possible to use neural networks to solve a broad range of financial problems.

The evolution of applying neural networks in business is described in the paper of Tkáč and Verner [97].

In one of the earliest works on this topic written by Coasts and Fant (1993), a comparison between multiple discriminant analysis (MDA) and neural network technologies was made [113]. The results confirm that neural network technologies have proven to be more efficient than the MDA-based model. Later, a paper by Lee and Choi (2013) [101] also confirms this result.

However, it is worth noting the disadvantages of this method. First, the neural network training is sensitive to the selected model parameters. Secondly, the construction of the neural network model and the interpretation of the results obtained is possible only if there is a sufficient amount of knowledge in this subject area. However, the neural network can give a reasonably accurate prediction with careful construction of the model: the choice of input variables and the selection of the necessary learning parameters.

Recurrent Neural Networks. The most popular method for performing classification and other analysis of data sequences is *Recurrent Neural Networks* (RNN) [114] . However, in problems of time series analysis, a modification of such networks is especially distinguished - *Long Short-Term Memory* (LSTM) networks [115].

The idea behind RNN is to use information consistently. In traditional neural networks, all inputs and outputs are assumed to be independent. But this is not suitable for many tasks [115]. For example, if it is necessary to predict the next value of the market index, it is best to consider the values that precede it. RNN is called recurrent because they perform the same task for each element of the sequence, depending on previous computations [116]. Another interpretation of RNNs is the networks with a 'memory' that considers prior information.

Pure recurrent neural networks are used sparingly in practice. The main reason for this is the *vanishing gradient problem* [114] . Ideally, for recurrent neural networks, a long chain of 'memories' is needed as an input so that the network can connect data relationships over significant distances in time [117]. For example, such a network could make real progress in understanding how events in the stock market are related. However, the more time steps we have, the more chances that backpropagation gradients will either pile up and explode or disappear.

To reduce the vanishing gradient problem and therefore allow recurrent neural networks to perform well in practice, there must be a way to reduce the multiplication of fewer than zero gradients [116]. A modified version of the RNN – the LSTM, is a specially designed logical unit that will help reduce the vanishing gradient problem enough to make recurrent neural networks more practical for long-term tasks [118]. It does this by creating an internal memory state added to the processed input signal, significantly reducing small gradients' multiplicative effect. In addition, the timing and impacts of previous inputs are controlled by a concept called the gate of forgetting. This concept determines which states are remembered or forgotten.

Hochreiter and Schmidhuber [119] introduced LSTM in 1997, and then optimised and popularised them in many subsequent works. Such networks do an excellent job of solving many problems and are widely used at this time. This method is considered to be one of the most powerful for market data analysis: [118], [120], [115], etc.

The benefits of using LSTM models over ARIMA and ARIMAX in financial data forecasting are multifold. Firstly, LSTM models can capture nonlinear relationships and complex patterns, providing a more flexible and accurate representation of the underlying data dynamics [121]. Secondly, LSTM models can handle long-term dependencies, making them suitable for forecasting tasks where historical information from distant past periods is crucial for accurate predictions. Lastly, LSTM models can effectively incorporate and leverage a wide range of input variables, including exogenous factors, enabling them to capture the impact of multiple relevant features on the target variable [122].

Decision Trees. *A decision tree* is a classification model that sequentially divides objects into subsets based on the value of the objects' attributes [123]. This model is a tree, the vertex of which corresponds to the entire set of objects from the domain.

At each vertex, a particular rule is specified with respect to one of the object's attributes and splits the set of objects corresponding to this vertex into several subsets (most often into two subsets), which in turn correspond to the child vertices of this vertex. The leaves of the tree also contain a label that matches all the objects associated with this leaf [124].

Having an object, to determine the class to which this object belongs following the model of decision trees, it is required to go down from the tree root to the leaf, answering at each vertex a question about some feature of the object and going down the corresponding edge of the tree.

The random forest classification method is based on creating a large number of decision trees and the final classification of the object based on the prediction of all the created decision trees [125]. Thus, the random forest averages the predictions of the decision trees. As a final prediction, the median of all predictions of the constructed decision trees is usually taken.

The procedure for constructing a set of decision trees from the initial training set is a core idea of the method. If we use the same data set and the same algorithm each time building the following decision tree, we get a set of identical trees.

Decisive trees of great depth are usually retrained on training data. They have a small bias of their predictions relative to correct predictions and have a considerable variance of their predictions [124]. The goal of a random forest is to reduce the variability in the prediction of individual decision trees.

One of the critical approaches used in the random forest method is *bootstrap aggregating* [126]. K data subsamples (with repetitions) are selected from the original training set, and a decision tree is trained on each of these subsamples. This approach allows reducing the variability of predictions without sacrificing bias, which is explained by the weak correlation values of each pair of decision trees in the set.

The second key point is that when constructing the following decision tree and deciding on selecting a feature for dividing the set of objects belonging to the next vertex into two subsets, a random subset of features is used. This approach allows further reducing the correlation between pairs of trees in the forest since it diminishes the number of occurrences of critical features that affect the final prediction at the vertices of decision trees.

The random forest method refers to non-linear classification methods, which complicates the interpretation of the results of this method [127].

Although the method of Random Forest is considered one of the most efficient for classification and features selection purposes, the resulting classification model is regarded as a "black box" [127] since it consists of a large number of decision trees of great depth, where each tree is trained on a specific subset of initial objects and signs.

2.6.3 Data Mining Software Applications Overview

The digital age has precipitated the proliferation of data from myriad sources, allowing businesses to leverage this information for strategic decision-making. Particularly in the financial sector, data mining has become an indispensable tool for actionable insights from complex datasets. Several software applications have been developed to aid this data mining process. This section critically evaluates six leading software applications for data mining related to financial and market data analysis.

RapidMiner. RapidMiner[5] stands out for its user-friendly data science platform and various functionalities. It supports various data mining techniques such as regression, clustering, and classification and features a visual programming interface that simplifies the model-building process. RapidMiner's predictive analytics capabilities make it particularly suitable for forecasting market trends, identifying profitable investment opportunities, and analysing customer behaviour. However, its major downside is that it can be resource-intensive, leading to slower performance on lower-end hardware.

IBM SPSS Modeler. BM SPSS Modeler[6] offers advanced analytics capabilities, demonstrating high robustness and versatility. It enables users to preprocess data, build and validate models, and deploy solutions efficiently. The SPSS Modeler excels in fraud detection, credit risk modelling, and customer segmentation in financial and market data analysis. However, the complexity of its interface and the steep learning curve might pose challenges for beginners.

Weka. Weka (Waikato Environment for Knowledge Analysis)[7] is an open-source software application that offers an extensive suite of tools for pre-processing, classification, regression, clustering, and association rules.

[5]`https://rapidminer.com/`
[6]`https://www.ibm.com/products/spss-modeler`
[7]`https://www.cs.waikato.ac.nz/ml/weka/`

Its simplicity and the availability of a wide range of machine-learning algorithms make it a popular choice among academic and commercial users. Weka has been used in finance for stock market prediction, portfolio optimisation, and trading strategy development. On the downside, Weka needs more scalability, which limits its application in dealing with very large datasets.

SAS Data Mining. SAS Data Mining[8] is an enterprise-grade application with sophisticated data mining and machine learning capabilities. It is a comprehensive and scalable platform for data manipulation, statistical analysis, and predictive modelling. It has been employed for credit risk analysis, fraud detection, and regulatory compliance in financial data analysis. However, SAS is generally considered more expensive than other solutions, which could be a barrier for some organisations.

KNIME. The Konstanz Information Miner (KNIME)[9] is an open-source, user-friendly, and comprehensive data analytics platform. It offers over 2000 modules for data access, transformation, model training, and visualisation. KNIME's intuitive graphical interface and extensive functionality suit financial market forecasting, customer behaviour analysis, and risk modelling. Although KNIME has many strengths, it is known for its relatively slower performance than some competitors.

Oracle Data Mining (ODM). Oracle Data Mining (ODM)[10] is part of the Oracle Advanced Analytics Database. It offers powerful data mining algorithms, allowing users to build, evaluate, and apply data mining models within the database, eliminating data movement and improving performance. In finance, it predicts customer behaviour, detects anomalies, and identifies key factors affecting financial performance. The primary disadvantage of ODM is its dependency on Oracle databases, which could limit its application in diverse database environments.

2.7 Python as a Machine Learning Engine Development Environment

Python has established itself as a premier language for machine learning (ML) and artificial intelligence (AI) owing to its simplicity, versatility, and interoperability with other languages. Its syntax clarity and powerful data processing capabilities make it an excellent candidate for ML tasks, which often involve collecting, organising, and analysing large volumes of data for subsequent algorithm development [128].

A noteworthy advantage of Python that bolsters its use in model development is its rich ecosystem of libraries and frameworks. These resources expedite coding, streamline model development, and enhance computation capabilities. NumPy, SciPy, and SciKit-Learn are integral libraries offering robust tools for scientific computation, advanced mathematics, and data mining. The aforementioned libraries are also compatible with machine learning frameworks such as Keras, TensorFlow, PyTorch, CNTK, and Apache Spark, further extending Python's utility in ML applications [129].

Python's intuitive syntax and high-level nature make it an apt language for teaching software development, especially in ML. This quality empowers developers to focus on solving intricate ML problems rather than wrestling with complex code implementations [128]. Moreover, Python's extensive support network and

[8] `https://www.sas.com/en_us/software/enterprise-miner.html`

[9] `https://www.knime.com/`

[10] `https://www.oracle.com/database/technologies/advanced-analytics/odm.html`

comprehensive documentation provide a wealth of resources for developers to reference during all stages of the development process.

A significant attribute of Python that bolsters its use in ML is its inherent flexibility. Python supports object-oriented programming and scripting, seamlessly facilitating the integration of various data types. This flexibility is especially beneficial when employing ontology frameworks for the creation of graph databases [130]. Furthermore, Python's cross-platform compatibility, which allows it to operate on a variety of operating systems, enhances its appeal to developers. The portability of Python code facilitates its deployment across multiple platforms with minimal modifications [130].

Beyond its computational prowess, Python also excels in data visualisation, thanks to libraries like Matplotlib. These tools empower data scientists to generate charts, graphs, and other visualisations, enhancing data comprehension, facilitating more engaging presentations, and simplifying user interface development [128].

Lastly, the thriving Python developer community has led to a substantial talent pool available for machine learning projects. This, combined with Python's simple syntax, has attracted many developers to contribute to ML endeavours, further solidifying Python's standing as a favoured language in the ML landscape.

Creating a Machine Learning Engine using Python environment. The NN code is unique for each modification of the model, depending on the user's requirements, data type. See Sections 5 and 6 for code examples for the two model modifications.

However, to create any of the NN modifications, approximately the same machine learning packages are involved.

NumPy[11] is one of the first libraries to be downloaded for Python, whether it is applying it for machine learning or fundamental data analysis. This Python machine-learning library provides capabilities for data and number manipulation. The library creates an N-dimensional array object into which users can place their data and offers functions for transforming this data.

The array object can also be easily adapted to other databases, making it natural for machine learning. In addition to being used in scientific and research applications, NumPy can create a container of shared data that can be easily manipulated. The library can solve problems related to linear algebra, transformations and random numbers.

NumPy builds on the weak computational power of Python by changing the data structure of an inline list into an n-dimensional array, which is the core functionality of NumPy. Processing data in this way provides more freedom with the data and allows the variety to contain heterogeneous data. This is extremely useful in machine learning applications, and the library's various functions for data cleansing are also essential to create a good machine learning algorithm.

Pandas[12] is one of the most integrated Python machine learning libraries a programmer should have. The library's name is an abbreviation for the term "data panel" and the module provides data analysis and

[11] `https://numpy.org`

[12] `https://pandas.pydata.org/docs/user_guide/index.html`

statistical functions. It also offers additional easy-to-use data structures to simplify data processing and preprocessing.

Pandas aims to add a lot of data analysis functionality to Python and are designed to enable users to perform complex operations without having to switch to a language for a more specific purpose. Pandas offers tools for reading and writing data to memory data structures, manipulating data with indexing, reshaping datasets, merging and merging datasets, filtering data, and time-series functionality.

The library is also highly optimised for performance since mission-critical code is written in Python or C. This optimises small operations while saving exponential time on more significant tasks. This, along with Pandas' powerful and functional statistical features, makes it the industry standard for data analysis in Python.

Statsmodels[13] is a powerful Python library designed specifically for statistics. Built on top of NumPy, SciPy, and matplotlib, it offers a wide range of tools for data exploration, statistical testing, and model generation. This library provides an array of regression models, such as Ordinary Least Squares (OLS), Generalised Linear Models (GLM), and Robust Linear Models (RLM), making it a versatile tool for understanding data relationships and predictions.

Beyond regression, Statsmodels excels in other statistical methods. It supports the Analysis of Variance (ANOVA), a standard statistical method for analysing differences among group means. The library is also proficient in Time Series Analysis, boasting a suite of models like Autoregressive (AR), Autoregressive Integrated Moving Average (ARIMA), and Vector Autoregression (VAR), making it apt for time-series data exploration.

However, it's important to note that despite its impressive functionality, Statsmodels does not cater to machine learning algorithms; other libraries like Scikit-Learn or TensorFlow are more suitable for these tasks. Nevertheless, with its robust set of statistical models, comprehensive testing functions, and detailed output, Statsmodels remains an indispensable tool for data scientists and statisticians aiming for deep statistical analysis.

TensorFlow[14] is a Python library created by Google in late 2015 for internal use in machine learning solutions. It was then open to the entire community and has since become one of the largest open-source Python libraries. The service is highly extensible and offers robust functionality for anyone looking to start with machine learning. The library uses a tiered node system that allows us to set up, train quickly, and deploy artificial neural networks with large datasets. This enables Google to identify objects in photos and understand spoken words in speech recognition applications. TensorFlow also contains necessary community tools and resources to ensure that users receive the best possible response continually. Even Google uses TensorFlow for its machine learning products like Gmail, YouTube, and Google Search. The ecosystem also has a repository of commonly used machine learning algorithms and models that can be used in plug-and-play mode.

While TensorFlow is used for classic machine learning, Keras is specially developed for deep learning.

[13]`https://www.statsmodels.org/`

[14]`https://www.tensorflow.org`

Keras is written from the ground up, focusing on providing library support for deep learning applications. It is designed to run on top of other libraries like TensorFlow, CNTK, or Theano.

Keras[15] was developed for a research project known as the Open Neuro-Electronic Intelligent Robot Operating System, or ONEIROS, and focuses on more straightforward prototyping, user-friendly interface, and modularity. It is more of an interface for building deep learning algorithms than a tool and supports convolutional and recurrent neural networks.

Keras is preferred for deep learning because of its easy-to-use interface and excellent support for parallelisation between CPUs and GPUs. The library also offers a modular iteration approach for deep learning. These modules can be infinitely customisable and connected to each other in any order to create a new deep learning solution.

Matplotlib[16] is one of the first Python machine learning libraries that a programmer should install as it performs an important part of a typical machine learning workflow. Visualisation. The Matplotlib library extends the functionality of Python by adding powerful visualisation tools. For example, the library allows us to create multiple graphs such as line charts, bar charts, scatter charts, and bar charts.

Matplotlib offers a MATLAB-like interface and makes it easy to visualise data. The Python backend allows programmers to extend the functionality of a library with toolboxes. Add-ons like Basemap, Cartopy, GTK tools, Mplot4d and Natgrid add much functionality to the Matplotlib library.

These features include map projections, Excel interface tools, 3D graphics, and image conversion capabilities. Matplotlib can be used to create 3D, image, contour, polar, and line plots, among others. This type of visualisation is necessary because it can help programmers identify patterns in data and speed up the imagination process. In addition, visualisation allows machine learning engineers to communicate their results in an accessible way accurately.

Scikit-learn[17] is a versatile Python machine learning library for many, thanks to its diverse use cases and powerful tools. Scikit-learn's primary goal is to offer powerful data analysis tools, with the library built on other powerful libraries such as NumPy, SciPy and matplotlib, with support for plotly, pandas and many more. SciKit offers functionality related to data classification and structuring. This also includes regression, clustering, dimensionality reduction, model selection, and data preprocessing. In addition to this, the library also has many commonly used machine learning algorithms, such as support vector machines, random forests, and gradient boosting.

This offers a lot of functionality, especially when combined with enterprise use cases for machine learning algorithms. This is why Scikit-learn is one of Python's most popular machine-learning libraries, used by nearly 40% projects on GitHub. It also ranks second behind TensorFlow in GitHub's best machine learning projects. SciKit is also used by companies such as J.P. Morgan, Spotify, and Evernote.

[15]`https://keras.io/guides/`
[16]`https://matplotlib.org`
[17]`https://scikit-learn.org/stable/`

3 A Generic Architecture in Software Engineering and the Process of its Development

This Section delivers a description of the process of computational modelling that will be taken as the basis for accomplishing the objectives of the research (see Section 1.2).

I relate this research to the area of experimental software engineering for data mining purposes, so it determines the choice of methodologies applied in the project.

3.1 Generic Component-Based Architecture

Generic Architecture. Under the *Generic Architecture (GA)* I mean an abstract framework with a generic structure with no physical realisation specification, i.e. its interpretation is not confined so far; it is a kind of template that defines the structure of the model and its data flow. It does not designate a specific domain usage, a type of data or a machine learning method which processes data.

This abstraction can be considered at different levels of detail. It allows the reuse of the model for a wide range of various projects with a relatively minor reorganisation [131].

The abstraction is immutable at the highest level but can be changed at the following (lower) levels depending on what prerequisites apply. For example, leaving all model components generic except for one determines how this model can be applied at the next level.

The following levels of abstraction in the context of the GA refer to the steps at which the model moves from generic to more domain-specific or concrete. This typically happens as one descends the layers of abstraction, from the highest level of abstraction to lower levels.

At the highest level of abstraction, the model is entirely generic, i.e., there are no specifics about the domain, data type, or machine learning method. The model is a ”template” for structuring data flows at this level.

Moving towards lower levels can be seen as a series of iterations where parts of the model become less abstract and more specific. This could be done by specifying, for example, the data mining area, the type of input data, or particular components of the model. Each specification narrows down the possible applications of the model, making it more domain-specific and less generic.

For instance, in the ga context, the next level of abstraction could be when one decides to use the model for a specific area of data mining or with a particular type of input data. The level after that could specify a particular machine-learning method to process the data. Every step taken from the highest level towards lower levels involves a move from a more abstract, less defined use of the model to a more concrete, well-defined one.

These levels of abstraction are not fixed and can be redefined or reorganised for different projects, allowing the model's reuse across a wide range of applications.

An excellent example of utilisation levels (layers) usage is NASA's Generic Software Architecture for Prognostics (GSAP) presented in [132]. The GSAP aims to apply prognostics technologies to diagnose a system and determine the evolution of a system's faults. It contains three layers – a framework layer (how to implement documentation), a deployment layer (including communicators, models, and prognoses), and a support layer (including support tools). The GSAP example, with its framework, deployment, and support layers, illustrates how these levels of abstraction could be structured.

Another example is the research of Wooldridge [105], which describes creating a system of intelligent agents. An agent is a computer entity that can consider the environment with the help of sensors and react to this environment with the help of actuators. Most intelligent systems are based on the concept of an agent. Different agents can be used depending on the system, e.g., simple reflex agents, utility-based agents, etc.

In [105], a system of agents that perform the decision-making function is provided as an *abstract architecture*. At the same time, a *concrete architecture* acts at a more detailed level and is aimed at different types of decision-making depending on the input domain. In our case, we can operate the GA as the abstract architecture and the PCM model dealing with the financial domain as a concrete one.

Component-Based Approach. The software system can be divided into abstract elements of the structure, such as components, blocks, or modules, using a component-oriented approach. These elements are designed to solve specific sub-tasks within the overall system task and interact with the entire system based on certain principles [133], [134].

The component-based approach requires having as many interrelations of the bits in a component with as little as possible between different components – this allows the use of the substitution of one component by another one [134].

Apart from the fact that the use of this approach makes it possible to create complex heterogeneous systems [134], it also simplifies the verification of the performance of both the system as a whole and individual units, additionally providing the ability to upgrade systems, e.g. to replace unreliable units or old components by an advanced version.

A model's component is a relatively independent system module, intended for reuse and deployment [133]. As can be seen from the definition, the use of component programming is intended to provide a simpler, faster and more straightforward procedure for computational model development, as well as to increase the percentage of code reuse, i.e. to strengthen the main benefits of an object-oriented approach.

It should be noted that the component-oriented approach is in some sense the addition to an object-oriented approach which is more suitable for developing large and distributed systems (for example, enterprise applications) [135].

Components are considerably larger units than objects. Among other differences between components and traditional objects is the ability to contain multiple classes and (in most cases) independence from a programming language.

The process of breaking a complex object into relatively independent parts is called *decomposition* [136]. When decomposing, it is considered that the connections between the individual parts should be weaker than the links of the elements inside the parts. At the same time, to assemble the developed object from the obtained parts, it is necessary to determine all kinds of connections between them.

The result of the decomposition is usually presented as a hierarchy diagram, at the lower level which relatively simple blocks are placed. At the upper level, an entire system is developed [135]. At each hierarchical level, the description of blocks is performed with a certain degree of detail, abstracting from irrelevant details [136]. Therefore, for each level, individual forms of documentation are used, reflecting the essence of the processes carried out by each unit. So, as a rule, it is possible to formulate only the most general requirements for an entire system. Plus, the blocks of the lower level must be specified so they can be assembled into a system.

In my case, I apply the decomposition method to divide the main models' complex functional tasks – data analysis & forecasting into a sequence of sub-tasks.

An analysis of several papers, including [137], [138] and [139], allows highlighting the following main characteristics of the components.

- *Reusability* – components are generally designed to be reusable in different applications. However, some components may be designed for a specific task.
- *Replaceability* – components can be freely replaced by other similar components.
- *Independence* – components are designed to work in different environments and contexts, having a minimum dependency on the other components.
- *Extensibility* – a component can be extended from existing components to provide new behaviour.
- *Encapsulation* – a component exposes interfaces that allow using its functionality and does not expose the details of internal processes or any internal variables or state.

Using Induction in developing Generic Architecture. Science is usually based on facts. So, first, facts are collected, compared, and conclusions are outlined; as a result, laws and patterns are formulated. The methods of obtaining these regularities are called methods of scientific research.

In my study, I use the method of induction. The basis of induction is a researcher's experience, experimentation and observation; by these means, individual facts are collected, allowing the researcher to identify common and recurring features of a certain number of phenomena belonging to a certain class [140]. Then, on this basis, an inductive conclusion is built based on statements about individual objects and phenomena with an indication of their recurring characteristics and a statement about the class that includes these objects and phenomena. Finally, a generalisation assertion can be obtained – characteristics identified for the set of unique objects are attributed to the entire class.

Inductive generalisations can be divided into the following types: popular, incomplete, complete, scientific, and mathematical induction [140]. In scientific research, incomplete induction is commonly used when a conclusion is made based on several facts related to the area under study. As the total number of facts impacting research can be incalculable, the experience can be considered infinite and incomplete; thus, the inductive conclusions are normally stated as probabilistic.

In my project, I primarily consider experimental modelling and, as a result, create a GA based on the facts obtained.

The first stage of the study is a detailed analysis of the problem raised and breaking it down into less complex parts. In the context of this thesis, the system's fundamental problem is set as *data analysis and forecasting using comprehensive data mining techniques.* This problem is directly related to the so-called *Knowledge Discovery in Databases* process.

Knowledge Discovery in Databases as the basis of the Generic Architecture. *Knowledge Discovery in Databases (KDD)* is a process of discovering valuable knowledge in databases. This knowledge can be represented in patterns, rules, forecasts, relationships between data elements, etc [141].

The primary knowledge search tool in the KDD process is Data Mining analytical technologies that implement the tasks of classification, clustering, regression, forecasting, prediction, etc.

Data Mining is a methodology of discovering previously unknown, non-trivial, practically valuable and accessible knowledge in large amounts of data that accumulate in the information systems, which is necessary for making decisions in various domains of expertise [2]. The knowledge discovered in the process of Data Mining is non-trivial and previously unknown. Non-triviality implies that simple visual or statistical analysis cannot discover such knowledge. The process allows for defining the complex relationships between the properties of the objects of analysis, predicting the values of some features based on others, and so on. Besides, it is supposed that knowledge received as an outcome of data mining should be applicable to new objects.

Knowledge discovered by the KDD process should be presented in a form understandable to users who do not have special mathematical training. For example, the logical constructions "if, then" are most easily perceived by a user. In the case when the extracted knowledge is not transparent to the user, there should be post-processing methods that allow them to be brought to an interpretable form.

However, following the concept of KDD, an effective knowledge search process is not limited to data mining. The KDD includes the sequence of operations required to support the analytical process [142]; these steps of the KDD are presented in Fig 3.

1. *Data Selection* – the process of extracting input data (including training data) from various sources (OLTP systems, DBMS, individual user files, Internet sources, etc.) and loading it into a system's data warehouse. At this stage, an initial dataset is generated. It should be large enough to contain sufficient information for the analysis to be extracted and, simultaneously, restrained to be used effectively.

2. *Data Pre-processing.* During this step, data is cleaned from factors that interfere with analysis (such as noise and anomalous values, duplicates, contradictions, omissions, fictitious values, etc.). Besides, the data can be structured, and visual and interactive statistical analysis can be performed (e.., by using data visualisation) to gain an understanding of data by detecting expected and unforeseen relationships between variables and deviations.

3. *Data Transformation* is the optimisation of data to solve a specific problem. Usually, at this stage, the

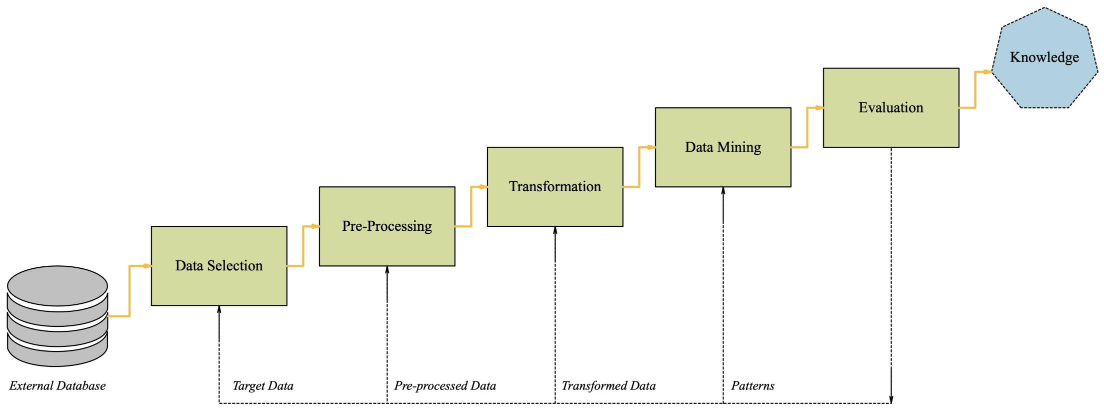


Figure 3: KDD process

elimination of insignificant factors, reduction of the input data dimension, normalisation, enrichment, and other transformations are performed to better ”adapt” the data to the solution of the analytical problem.

4. *Data Mining* - application of Data Mining methods and technologies: building and training models (neural networks, decision trees, etc.), solving the particular problems of data analysis (classification and regression, clustering, forecasting, searching for associations, etc.) Interpretation and visualisation of analysis results.

5. *Evaluation.* This final stage consists of the comparison of the results with the planned indicators or already known information and the analysis of the reliability and usefulness of the created models.

Knowledge Discovery in Databases does not specify a set of specific processing methods or algorithms suitable for analysis; it defines a sequence of actions that must be performed in order to obtain knowledge from the source [143], [144]. This approach is universal and does not depend on the subject area, which is its undoubted advantage.

In this project, the KDD standard is used as the basis for the development of the GA for PCM model applying the new semantic methods of information research and modification.

Following the KDD standard, I divide the process of performing a complex fundamental problem of analysing and forecasting data into the following *stand-alone sub-problems* that will form the blocks of the PCM model: preparation of raw input data(1), structuring and storing data (2), feature selecting data for further analysis & formation of the new more efficient dataset (3), data analysis itself (4), data evaluation and results’ improvement (5).

To improve the system’s quality, searching for optimal tools for solving each sub-problems is necessary. Therefore, these sub-tasks are considered separately at the initial stages of the research.

Next, I should analyse the complex problem from a modelling perspective and build PCM.

3.2 Computational Model

The term *Computational Model* does not have a singular interpretation in literature; in my work, by a computational model, I mean a set of interconnected and interacting computational procedures and methods designed to collect, store, process and distribute information.

According to studies, including [145], [146] and [147], the development of a computational model pursues the following primary purposes: increasing system performance by accelerating data processing, improving the reliability of calculations, providing users with additional services, etc.

One of the well-known examples of applying the computational models for machine learning data analysis is a probabilistic graphical model based on Bayesian inference [148].

One of the main tasks in developing a computational model is to define a set of its underlying methods [147]. I propose combining machine learning methods and semantic data preprocessing to solve particular data analysis problems, e.g. data classification and prediction.

The PCM model can be considered unique in the literature due to its comprehensive nature and the integration of multiple components that collectively contribute to its distinctiveness. The term *"comprehensive"* in this context refers to the model's ability to cover various aspects of data analysis holistically due to the incorporation of different components and techniques to address the complexities of financial data.

Overall, the comprehensive nature of the PCM model arises from its integration of information management and decision-making, semantic and graph-based methods, emphasis on advanced data pre-processing, adaptability to various domains and data types, and continuous improvement through a feedback loop.

Only a limited number of publications propose a unified methodology that emphasises integrating the semantic approach with machine learning. For instance, papers [149], [150] discuss applying such an approach in biomedical studies. Similarly, paper [151] presents a comparable methodology for addressing text mining issues. Furthermore, [152] suggests the utilisation of ontology and machine learning within civil engineering. However, it's essential to acknowledge that all these papers are published no later than the year 2022.

Thus, further development of the comprehensive computational model regards a combination of a set of separate tools: ontology, graph database, and machine learning techniques.

Computational Model Development Cycle. The analysis of a number of articles e.g. [153], allows us to conclude that the process of creating any *single experimental computational model* in software engineering can be divided into six stages:

1. *Task definition*, i.e. definition of the simulation object. At this planning stage, information is collected, the formulating of the question, the definition of goals, and the description of the data.

2. *System analysis and research*, i.e. analysis of the system, a detailed description of the object, development of an information model, analysis of technical and software tools, and development of a mathematical model.

3. *Formalisation*, the creation of an algorithm, i.e. choosing the algorithm design method, the algorithm recording form, the testing method, and the design of the algorithm.

4. *Programming (physical realisation)*, i.e. choosing the programming environment or application environment for modelling, refinement of the methods of data organisation, recording of the algorithm in the chosen programming language (or in the application environment).

5. Carrying out a series of computational experiments, *testing*, i.e. debugging of syntax, semantics and logical structure, test calculations, adjustment of the program.

6. *Analysis and interpretation of results.* The ultimate goal of modelling is to make a decision that must be worked out on the basis of a comprehensive analysis of the results obtained. This stage is decisive - either we continue the research or finish it. If the result is known, then it can be compared with the modelling result.

It should be noted that the simulation is a cyclic process. This means that the first six-step cycle can be followed by a second, third, etc. At the same time, knowledge about the investigated object is expanded and refined, and the initial model is gradually being improved. Disadvantages found after the first modelling cycle, due to little knowledge of the object or errors during the designing of the model, can be corrected in subsequent cycles.

I employed the model development cycle methodology when working on the two use cases (see Sections 4 and 5).

Through a series of experiments, I build a set of similar models. When developing each model, I declare the features and similarities of these models, determine the functionality of the blocks, and the data format for transfer between blocks. For example, it is crucial to ensure that one block "talks" to another and, thus, information is transferred from one block to another correctly.

After a series of successful experiments, I can move on to formulating a standard unified model and framework for it.

Since I want this model, which in a broad sense is responsible for the analysis and prediction of data, to be realisable for other use cases, it is necessary to create an abstract architecture. The architecture's solutions to the previously analysed sub-problems become its components.

In a rapidly changing requirements and unstable environment, introducing or developing software parts with constant tracking of the finished product and continuous communication between the developers is much more effective than consistently implementing programs with large amounts of documentation - a traditional approach. Thus, if the project is subject to some internal or external uncertainties, it is reasonable to plan the computational models' life cycles using the Agile methodology.

Agile methodology. According to Agile software development methodology, tasks are divided into small parts, called *iterations*, with careful short-term planning and approximate long-term planning. For the classical iterative model the iteration length can be arbitrarily large (but within reasonable limits), meanwhile, for the Agile approach the time is limited and allocated for the execution of a concrete iteration (up to five weeks) [154].

Agile is based on empirical management, i.e. decisions are made based on the intermediate results of the project. Also, these results are transparent, which means that all people involved in the project are aware of the status of the project, the number of changes and possible problems [155]. It should be noted that in the case of IT projects, empirical management allows quickly making adjustments to the software product, and this is a significant advantage over the waterfall model of the life cycle. Thus, this method is suitable for developing a computational model within this project.

3.3 Process of Developing a Generic Architecture

In this Section, the following steps of creating a GA for PCM model are formulated, which are also shown in Fig. 4.

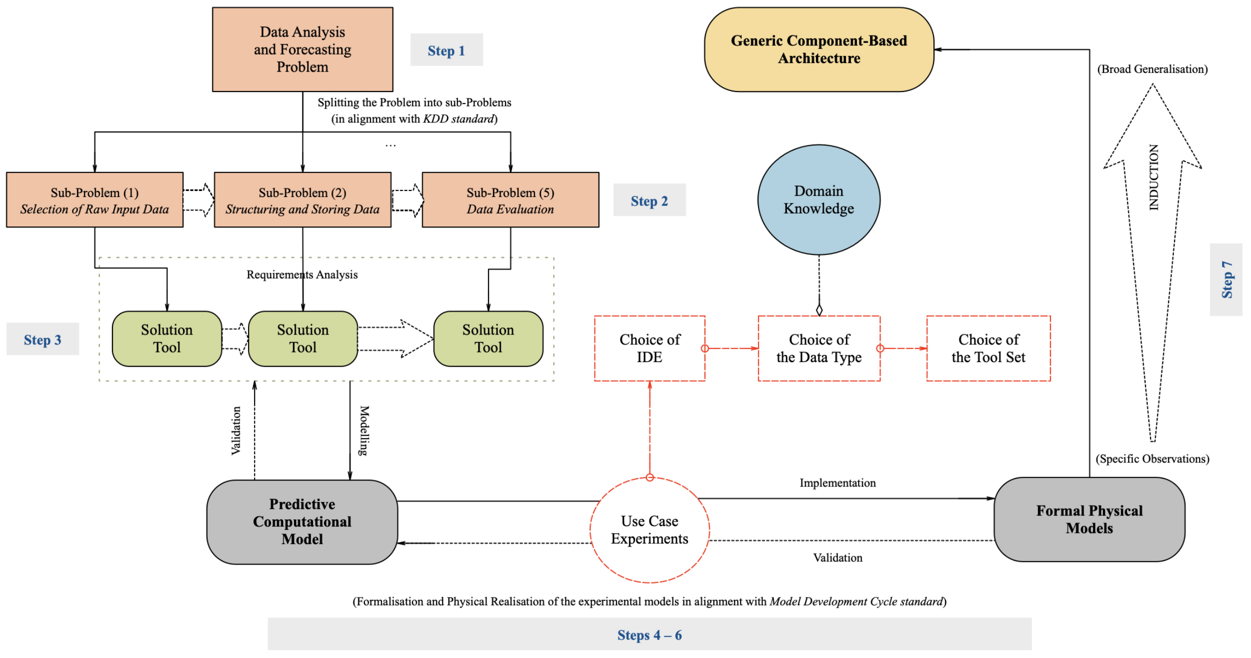

Data Analysis and Forecasting Problem
Step 1
Generic Component-Based Architecture
Splitting the Problem into sub-Problems (in alignment with *KDD standard*)
(Broad Generalisation)
...
Sub-Problem (1) *Selection of Raw Input Data*
Sub-Problem (2) *Structuring and Storing Data*
Sub-Problem (5) *Data Evaluation*
Step 2
Domain Knowledge
INDUCTION
Requirements Analysis
Step 7
Step 3
Solution Tool
Solution Tool
Solution Tool
Choice of IDE
Choice of the Data Type
Choice of the Tool Set
Validation
Modelling
(Specific Observations)
Implementation
Predictive Computational Model
Use Case Experiments
Formal Physical Models
Validation
(Formalisation and Physical Realisation of the experimental models in alignment with *Model Development Cycle standard*)
Steps 4 – 6

Figure 4: Generic Architecture Development Process

Step 1. Understand the Complexity: The first step involves acquiring a comprehensive understanding of the Data Mining and Forecasting problem. This process includes segmenting the multifaceted issue into more digestible and manageable parts, aligning with the KDD process.

Step 2. Form Model Components: The derived sub-problems serve as individual components (or blocks) of the model:

Sub-Problem 1: Raw Input Data Selection

Sub-Problem 2: Data Structuring (Pre-processing) and Storage

Sub-Problem 3: Feature Selection for Analysis and Creation of an Optimized Dataset

Sub-Problem 4: Application of Data Mining Tools for Pattern Extraction

Sub-Problem 5: Data Evaluation and Result Refinement

Step 3. Select Optimal Tools (Methods): This step involves identifying the most effective tools to solve each sub-problem. These solutions then serve as distinct blocks within the model. For instance, to address *Sub-Problem 2*, we can leverage the Graph Database methodology for data structuring and storage.

Step 4. Develop Strategy: In this phase, we scrutinise the overarching Data Mining and Forecasting problem from a computational perspective. The goal is to formulate a strategic blueprint for constructing the Predictive Computational Model, adhering to the principles of the Computational Model Development Cycle (see Section 3.2).

Step 5. Construct Experimental Models: A series of experimental models are created. Fig. 4 shows the detailed process of experiments' validation. These experimental models would later become use cases once the model is fully validated.

Step 6. Identify Model Features and Data Flow: The next step is to discern and declare these experimental models' commonalities and unique features. This process also includes defining the functionality of each block and how data traverses between them.

Step 7. Formulate Generic Architecture: The concluding step consolidates the components into a standardised, unified Predictive Computational Model, forming the Generic Architecture. The tools to the sub-problems outlined in *Step 3* become integral components of the GA.

3.4 Generic Architecture of Predictive Computational Model

The generic component-based architecture for the *Predictive Computational Model (PCM)* is shown in Fig. 5.

It consists of the following components: *Raw Data Interpreter*; *Ontology*; *Graph Database*; *Feature Selection Component*; and *Machine Learning Engine*.

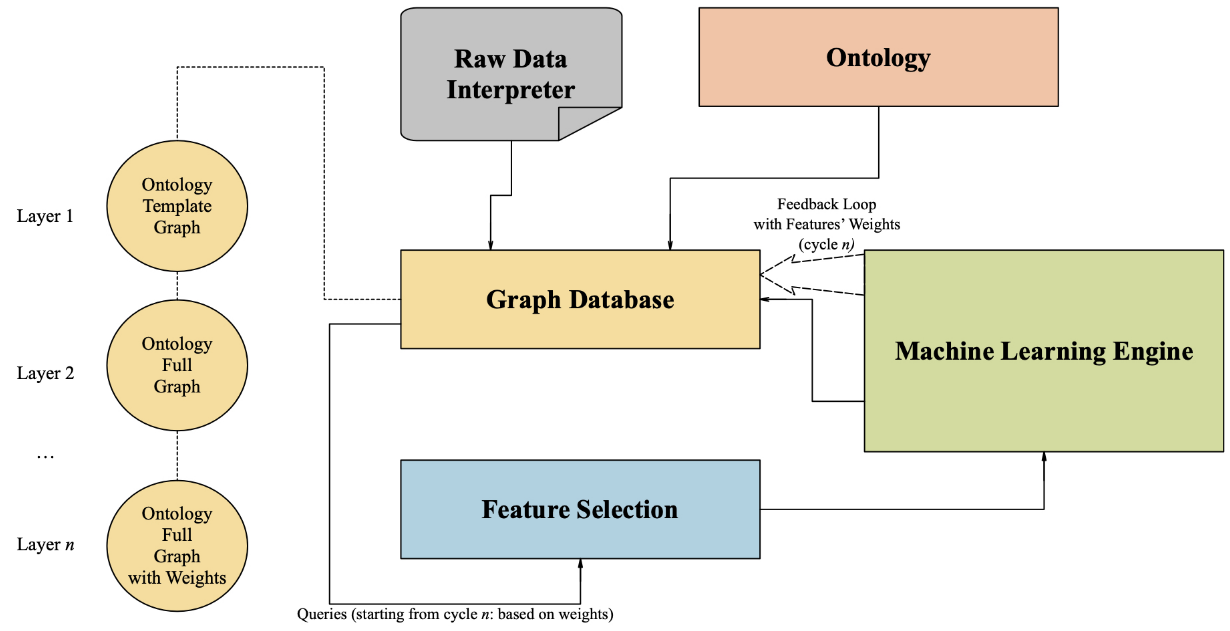


Figure 5: Generic Architecture for the PCM model

Raw Data Interpreter. The role of this module is to collect raw data from the source' standard database. The data gained are subsequently stored in a convenient form to support its efficient management. The interpreter allows feeding the data in an appropriate commonly used format (e.g., CSV) to the Graph Database module (see below).

As the model I propose should be *generic*, it can be applied for data mining of all kinds of quantitative datasets; however, it is more suitable for highly interconnected data. As it was mentioned before, for my use cases (see Sections 4 and 5), I have determined to use financial/market data.

Semantic Database System. Under *Semantic Database System (Semantic DS)* I mean a combination of an Ontology, Graph Database and Feature Selection, which allows using both experts' knowledge about the particular analytics field collected in standardised ontology format and input datasets exported and structured in Graph Database for further effective data pre-processing. It is a vital part of the model, which ensures data pre-processing efficiency by selecting, structuring, and giving meaning to raw data. My colleagues and I describe the advantages of using such a method in our paper [1] and Section 2.3.

The decision to include Ontology in the system is due to the need to formalise the theoretical concepts needed for the data analysis of a particular type of data and create a framework according to which the entire system will operate.

A system's ontology defines a standard concept library for users and developers who need to share information in a domain. It includes machine-interpreted formulations of the basic concepts of the subject area and the relationship between them, making them more accessible for complex reasoning.

Graph Database (Graph DB) is the system's local database which stores the raw data exported from the external sources database, structuring it in a form of an interconnected graph based on the information taken from the Ontology. By means of Graph DB, the data can be adapted to the further system's components – Feature Selection and Machine Learning Engine. Additionally, Graph DB is dependent/dynamically updated by the feedback from the Machine Learning Engine (the core component of the system which returns the analytic outcomes). It allows Graph DB to keep the outcomes and analytical metadata (e.g. features' weights) from the models' previous iterations.

Using a Graph DB instead of the traditional one is mainly due to the fact that it copes well with highly interconnected data (e.g., financial data). It effectively implements the property graph model, right down to the storage layer. This means that the data is stored exactly as the user represents it. Only a database that initially contains all possible connections can efficiently store, process, and query relations. While other databases compute relationships at query time through resource-intensive operations, a graph database stores connections and the model's data. Time traversal constants on large graphs facilitate the efficient representation of nodes and relationships. Therefore, regardless of the overall size of the dataset, graph databases do an excellent job of managing data with complex relationships and complex queries. Moreover, the flexible property graph layout can adapt over time, allowing to subsequently materialise and add new relationships to quickly access and speed up domain data as your analysis needs to change.

I use *Protege* environment for ontology construction and *Neo4j* environment for Graph DB construction in my use cases. However, according to the component-based approach, these components can be substituted with the other ones with the same functionality which are more suitable for the particular user/ developer.

Both tools, ontology and graph database, are "compatible" with popular programming languages, including Java, JavaScript, .NET, Python, etc. For example, data from ontology (e.g., RDF format) can be easily imported to a graph database using Python code. This help to automate and speed up the system management and construction process and integrate these independent components into a single system.

To make my approach mathematically rigorous, I need a formal definition of a graph-based ontology representation. However, here I should aim to introduce theoretical concepts which are amenable to practical implementation. My solution here is to define the ontology representation in the form of a *Labelled Graph*. My intuition here is as follows – nodes and edges of the graph structure would reflect the ontology structure while I will use a dedicated language to label graph nodes. In my case company's features are presented as nodes, and the relationships between these features are presented as the edges. Informally, labels will allow us 'to keep track of the data management process' – labels evolve from just an 'abstract container' for specific metadata (e.g. features' values and weights) to those containing concrete data. Subsequently, below I define, three relevant concepts of such labelled structures – Ontology template graph (definition 3.3), where nodes are labelled by these 'abstract containers'; an Ontology full-graph (definition 3.2), where these 'abstract containers' are filled with the values gained from the concrete financial data and OBP Ontology full graph with weights (definition 3.1), where after the Machine Learning Engine feedback loop adds the weights of the features.

Definition 3.1 (Ontology full graph with weights)**.** An *Ontology full graph with weights* is a labelled graph

$$G_w = \quad < V, E, L_w >,$$

where V is the set of vertices, E is the set of edges (features), and L is the set of labels. Labels in

$$L = \quad value : 0, \quad value : i, \quad weight : 0, \quad weight : k$$

where $value : 0$ is a constant meaning 'the value is not yet identified' and i ranges over the real values taken from a source dataset; $weight : 0$ is a constant meaning 'the feature is irrelevant regarding this particular model's cycle' and k ranges over the weights which were added as the result of analysis executed during the previous model's cycle, besides k ranges in [0,1], where 1 means there is a direct dependence between the outcome of the model and the feature's value.

Definition 3.2 (Ontology full graph)**.** An *Ontology full graph* is a labelled graph

$$G = \quad < V, E, L >,$$

where V is the set of vertices, E is the set of edges (features), and L is the set of labels. Labels in

$$L = \quad value : 0, \quad value : i, \quad weight : o$$

where $value : 0$ is a constant, meaning 'the value is not yet identified' and i ranges over the real values taken from a source dataset; $weight : 0$ is a constant meaning 'the feature's weight is not yet identified'.

Definition 3.3 (Ontology template graph)**.** An *Ontology template graph* is a labelled graph which, as the full graph, contains a full set of vertices, edges, and labels exported from the ontology module; however, the values in labels are not identified. In other words, it's a graph which contains no information about a particular object of analysis.

When the template graph containing the core ontology information is created, it should be filled with the data - in this way, we transform a template graph into the full graph. After the first iteration of the model, the full graph is supplemented by the weights of the input features.

Thus, these three types of ontology graphs are layers of the graph database system.

The attributes used to train the model significantly impact the quality of the results. Uninformative or poorly informative features may reduce the effectiveness of the model. Therefore, the process of selecting features that have the closest relationship with the target variables is performed. During each iteration, features are corrected, considering the models' previous iterations.

Feature Selection (as well as Data Structuring, Filtering and Visualisation) is optimised by Graph DB queries. Using the weights (importance coefficient) of the features calculated in the previous cycle of the model, the most valuable features can be selected using complex queries. The graph database allows generating the new input subset of features for the Machine Learning Engine (the next model's cycle).

Graph databases enable the formulation of the queries of any complexity handled by a dedicated query language (e.g., Cypher as part of Neo4j in my case). The efficiency of queries, though, is supposed to be improved by an expert.

Machine Learning Engine. Finally, data analysis & forecasting using selected key features can be carried out through *Machine Learning Engine (ML Engine)* , which includes a set of Data Analysis methods, such as Neural Networks, Regressions, Decision Trees, Support Vector Machines, etc.

The ML Engine method choice depends on the particular input dataset characteristics and the specific nature of the analytical objectives (forecasting or classification). For example, for one of the variations of PCM – *Bankruptcy Prediction Computational Model (BPCM)* (see Use Case 1, Section 4), where input features are a company's financial ratios, I use Classical Neural Network to identify which category a company relates to (bankrupt or non-bankrupt).

For *Market Index Prediction Computational Model* (see Use Case 2, Section 5), the input parameters consisted of the time-series of FTSE100 close prices and Market Price Index and Macroeconomic Indicators historical values, so I applied a Hybrid ML Engine, which is a more complex one, combining a bunch of Long Short Term Memory networks plus the LR (see Section 5.4).

Feedback Loop. Besides returning the outcome of the current iteration, ML Engine allows analysing the weights of the input features and transfers this information to the Graph DB component by the feedback loop, enhancing the system performance during the subsequent iterations.

In Fig. 6, the Feedback Loop is shown separately from the computational model.

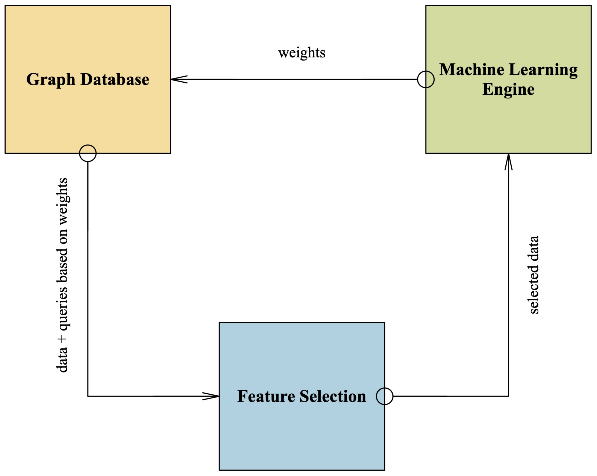


Figure 6: PCM Feedback Loop Architecture

Thus, the system represents a dynamic recurrent process of n iterations (cycles) that continually researches two central questions: how the provided features impact the target variables and what analytical conclusion the system produced for the particular iteration.

Condition for PCM Generic Architecture. I state that my GA works under the following conditions.

First, before starting with the model realisation, the primary *Integrated Development Environment (IDE)* for developing the PCM model components and the links between them should be chosen. In my use cases, I used *Python IDE*, as, by the moment of the project research, it was verified that it contained all the necessary instruments (libraries), allowing us to implement the model.

The raw dataset should contain quantitative heterogeneous data presented as a data frame. Here, by *Data Frame* , I mean "a 2-dimensional labelled data structure with columns of potentially different types"[18]. Besides, the raw dataset should be saved in a format that enables data passing from one model component to another, for example, in CSV or similar.

A model allows using a third-party Ontology, or the user can develop it if they have enough domain expertise. Besides, the Ontology should be presented in the standard RDF or OWL format.

The Ontology should contain a formal description of all the concepts related to the potential raw dataset. All the possible titles of the features potentially present in the Raw Dataset should be presented in the ontology as the concepts (as the main titles or synonym titles). The Ontology should be checked for the meta-data content required for the potential raw dataset, e.g. annotations containing non-hierarchical relationships with other concepts, synonyms, data properties, etc.

The choice of the feature selection queries and Machine Learning Engine content depends on the nature of the raw dataset.

It is important to note that the listed conditions are not related to *the efficiency* of the model.

Going down to the next level of the abstract model means it is saturated with more explicit content. For example, a dataset type is defined (e.g., financial), which has differences from other similar data types (e.g., security data, medical data, etc.), so I go down one level of abstraction below. Further, the types of datasets can be divided into subtypes; thus, getting more specific, the model goes down to the next level.

Since the model needs to be filled with specific data to ensure its effectiveness, it was decided to use *financial /market data* as input.

Semantic Approach for Financial Data Mining within the Framework of PCM. I argue that the financial datasets to be analysed for my purposes, figuratively speaking, can be characterised by *four 'V' and 'R'.* It shares most (four out of five big 'V') of the qualities of *Big Data* – Variety, Velocity, Veracity and Value [156] being not dependent on the Volume. However, I underline the fifth, 'R', a feature of financial data – an extremely high level of Relationships (for justification, see Section 2.3).

Although traditional *Relational Databases* still dominate among data storage facilities, these systems would not be suitable for the purposes of financial analysis, being unable to tackle the requirements of the

[18]`https://pandas.pydata.org/docs/user_guide/dsintro.html`

'Big Four V + R'.

There are *NoSQL systems* that extend the capabilities of traditional databases by allowing them to deal with the four 'V' – Value. It would have been possible to utilise solutions developed for Big Data management, such as Scribe, HBase, Cassandra, etc. [157]. However, it would bring unnecessary complications as these solutions have been developed to tackle the Volume of the Big Data, a feature that our target data would not have. Moreover, these solutions were not designed to tackle the 'R' feature of our datasets.

While the *Value* feature with respect to Big Data, normally denotes the worthiness, usefulness of the data (a pragmatic meaning), I bring here the semantic approach as a specific framework in which the desired usefulness will be tackled.

I argue that the range of problems related to integrating data can be resolved within the framework of a semantic approach to data modelling. The *Semantic Approach* in our case is used to measure the connotative content of information (meta-data), i.e. it makes information retrieval more accurate and relevant to the query [31].

I believe that representing information in a graph benefits this approach. Indeed, when objects are correlated with the nodes of a graph and their relationships are associated with the edges, it is possible to effectively add data from different sources into one structure. The graph structure is the most convenient for representing complex engineering information when I need to constantly develop and maintain the data model throughout its life cycle [158]. Information in a semantic form is quickly rebuilt and expanded when new sources become available, without the need for primary processing of the storage system, as in the case of traditional databases. The semantic approach supports flexible and extensible information models, allowing to a combination of financial information and avoiding the termination of the editing of the information model with each iteration of extension. The format and volume of the processed data can be specified as information requirements evolve and the knowledge of the needs of the participants in the life cycle develops.

Additional advantages of the semantic approach are provided by *ontological standards*, which allows not only obtaining information from different sources in one flexible and expandable format but also interpreting it in the same way [159].

Further, the semantics of the obtained data can be clarified from previously known sources (reference data libraries) using the same standardised technologies and tools used for data exchange. The combination of semantic and ontological standards helps to organise the exchange and comparison of data, the identification of conflicts and the harmonisation of contradictions. Semantic and ontological standards are also suitable for data processing tools of varying degrees of data complexity. Besides, publicly accessible digital data storage specification called *open data format* is free from licensing restrictions [160].

Creating an Ontology for PCM in Protégé environment The ontology for PCM is created in two stages: an informal conceptual catalogue of all the terms (objects) – this is the task for experts in the specific domain field, and a logical-semantic functional model in Protege based on a conceptual catalogue developed by experts using the resource metadata description language. In my case, it's Resource Description Framework (RDF format).

The basic element of OWL/RDF is the class of all classes, defined as *owl:Class.* Each OWL class is a child of the *owl:Thing* class and a parent of the *owl:Nothing* class. Usually, it is enough to declare a certain entity as an instance of a certain class so that this individual automatically becomes an element of the *owl:ThingClass.*

Class inheritance in the OWL language is specified using the RDF *subClassOf* construct. According to RDF, the fact that one class is a child of another means that all instances of the child class are instances of the parent class.

After the hierarchical structure of the ontology is built and all the classes and subclasses are added, I can proceed to a more detailed description of each element of the ontology.

OWL allows annotations with various information and metadata for classes, properties, individuals, and header ontologies. These pieces of information can are instrumental as editorial information. For example, it may include comments, creation date, author, links to resources such as web pages, etc. OWL Full does not put any restrictions on the use of the annotation property.

OWL has five predefined annotation properties that can be used to annotate classes properties and individuals. For example, they include:

- *rdfs:Label* (a literal field), which can be used to add explanations to a person/computer when reading the names of elements such as classes, properties, and individuals in the ontology. I use this annotation to add the synonyms of elements' main titles. It can also be used to provide multilingual names for ontology elements.
- *rdfs:comment* (a literal field) is used by us to add the terms' definitions with sources.
- *rdfs:seeAlso* (URI reference) that can be used to refer to the other classes connected with the given class.

It is also allowed to add additional annotations. I use this option to define the non-hierarchical connections between elements. For example, for BPCM model ontology, I created the special annotation called *rdfs:ratioRelationships* with to subclasses – *directlyRelatedTo* and *indirectlyRelatedTo.* When adding the new annotation to a particular element, I use the URI field to refer to the dependent elements from another Class. This is how the connections are built.

In OWL, there is a separation of properties into two classes:

- Object properties are used to bind individuals to each other. Object properties are instances of the *owl:ObjectProperty* class
- Data type properties associate individuals with what are called data values. Values here are RDF literals or data types defined in XML Schema. Data type properties are instances of the *owl:DatatypeProperty* class.

I use the *owl:DatatypeProperty* to define the "empty containers" for the model's input data to be filled in (e.g. 'Value', 'Weight') after the Ontology will be transferred to Neo4j environments.

The *owl:Object Property* and *owl:DatatypeProperty* classes are children of the *rdf:PropertyClass*. OWL allows to impose restrictions on properties by setting the axiom that these properties must satisfy. Some types of axioms in OWL are named and there are special language constructs for their definition.

OWL language allows describing various characteristics of classes and properties, which are usually set as various restrictions on the structure of relationships between their instances. These restrictions are expressed as predefined relationships, called axioms in OWL.

The axioms of classes define the rules on the basis of which class instances communicate with each other. The main axiom of a class is the declaration of its name as the name of the class. This axiom suggests that a class with that name exists. But this, of course, is often not enough to express even simple hierarchical relations between classes. Therefore, in the OWL language, three more class axioms are distinguished:

- Axiom *rdfs:subClassOf*, which describes a class as a subclass of another. As mentioned above, if a class is a subclass of another, all its instances are also instances of the parent class
- The axiom of equivalence of *owl:equivalentClass*, which allows us to say that the sets of instances of these classes coincide. Usually, this is just a way of specifying synonyms.
- Opposite in the sense of the previous axiom of equivalence, axiom *owl:disjointWith*. This axiom is a way of saying that sets of class instances have no common elements.

When the ontology is ready, it should be saved as an RDF file for transferring it to the next model's component.

The Protégé RDF file can be easily managed and read in the Python environment using the owlready2 library. I will use this when transferring the file to the Neo4j environment.

Creating a Semantic Graph Database for PCM in Neo4j environment Cypher query language was originally developed specifically for the Neo4j graph database environment. Cypher's goal is to provide a human-readable SQL-like query language for graph databases. Several graph databases currently support Cypher. The openCypher[19] organisation was created to standardise it.

A brief manual on creating a graph in the Graph Database using Cypher is provided in Appendix A.

However, for PCM Model I don't need to create the Neo4j graph from scratch. It will be created automatically by means of a Python environment. On the basis of the Ontology file, I created in Protege.

To do this I need to use the following packages for Python: neo4j, owlready2, in addition to pandas and numpy. Package *owlready2*[20] helps read and manage the RDF file and transfer the particular information from it to other platforms. Package *neo4j*[21] allows to launch the Neo4j graph from Python and manage it there by means of Cypher queries.

First, I need to open the created empty graph in the Neo4j app. Then define the LAN connection parameters to connect to the graph database in Python.

[19] `https://opencypher.org/`

[20] `https://owlready2.readthedocs.io/en/latest/`

[21] `https://neo4j.com/developer/python/`

Then a separate file is created in the Python environment, into which the code for transportation will be placed. Now I need to import the LSTM file itself (the owlready2 library will help us with this). After that, I can proceed directly to extracting information from the RDF ontology and using it as a base for building a graph.

Nodes and Relationships are created in Python as well as in the environment itself using the Cypher language.

All elements of the ontology are transferred as graph nodes. In this case, it is necessary to get rid of all prefixes in the names of entities, which are automatically added to the RDF file. Next, the data properties "containers" for each node (like Value, Weight, etc.) should be created. Then hierarchical relationships between entities are added, and then the relationships that are hidden in the annotations of the ontology (non-hierarchical). Thus, an *Ontology template graph* is formed in the environment (see Definition 3.3).

To convert an Ontology template graph to an *Ontology full graph* (see Definition 3.2), a graph that is already filled with data, I need to use data from a CSV file with data taken from an external source. So the first step is to export the file itself to the Python environment and then correlate the names of the created nodes with the names of the columns in the file.

To save the necessary data from the Graph DB for transferring to other model components, a separate new CSV file should be generated.

After the first cycle of the model, the system generates a CSV file with the weights of input indicators (created using ML Engine). After this moment I can proceed with Feature Selection.

Feature Selection as the Part Graph Database Component In my PCM model, I need only those Feature Selection methods which are able to output a weight (or importance ranking coefficient) vector. Although the calculation of the weight values will be held in Machine Learning Engine, the outcomes are transferred to the Graph Database, where the new subset of selected features for the next models' cycle is formed (by means of Feedback Loop). Thus, the optimal choice of Feature Selection operators for these purposes are Regression (Lasso, Linear, etc.), Support Vector Machine (with the linear kernel), Gradient Boost Tree, and similar others.

When the weights are exported to the Graph Database after the first cycle, the *Ontology Full Graph with Weights* (see Definition 3.1) layer of the Graph DB can be created.

Next, based on the weights obtained, I can go directly to the Feature Selection by means of database queries.

From several indicators in one category, one with the *maximum weight* is selected. At the same time, Neo4j should display a graph containing only the indicators remaining after the selection. The data of this graph is exported as a CSV file for transporting it to the next component of the model.

4 Use Case 1: The Implementation of Bankruptcy Prediction Computational Model

In this Section, I will describe how to use PCM for the purposes of companies bankruptcy detection.

Bankruptcy Prediction Computational Model (BPCM) is the use case of PCM, which makes an analytical conclusion regarding the bankruptcy status of middle-size UK companies.

The architecture of BPCM is presented in Fig. 7.

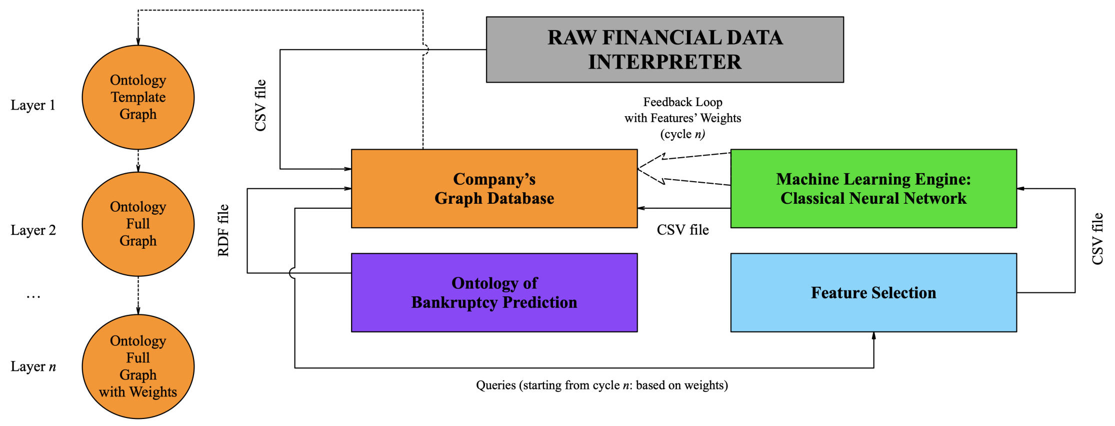

RAW FINANCIAL DATA INTERPRETER
Layer 1
Ontology Template Graph
Layer 2
Ontology Full Graph
...
Layer n
Ontology Full Graph with Weights
CSV file
RDF file
Feedback Loop with Features' Weights (cycle n)
Company's Graph Database
Machine Learning Engine: Classical Neural Network
CSV file
Ontology of Bankruptcy Prediction
Feature Selection
CSV file
Queries (starting from cycle n: based on weights)

Figure 7: BPCM Architecture

4.1 Input Data for Bankruptcy Prediction Computational Model

The first stage of model development is selecting a number of variables that accurately and fully describe the specific activities of the companies being evaluated.

Initially, most studies used variables that were selected on the basis of prevalence and popularity in the literature, e.g. Beaver used two ratios for his model - 'Working Capital to Debt' ratio and 'Income to Total Assets' ratio [73]. Until Altman [74] started with a rather large set of analytical ratios (22) and then reduced them to a minimum with the help of the MDA (5).

Currently, dozens of performance indicators predict companies' financial state. The following groups can be identified: liquidity, financial stability, business activity, profitability [161]. So the financial analysis is considered to be fulfilled if at least one performance indicator from each group is considered.

In this project step, I consider using 14 quantitative financial ratios, which were picked following several experts' advice. They are listed in Table 1. All of them are included in the list of the most popular financial indicators ever used for bankruptcy prediction purposes mentioned in Bellovary et al. review [162].

It should be noted that I picked the introduced ratios as they are suitable for all kinds of businesses, which is justified by IFRS Standards[22] and the UK Companies Act 2006[23]. Although, depending on the model user's demands and a particular company focus, this list can be supplemented by other more specific ratios.

In the future, some qualitative variables can extend the input ratios list (for example, sentiment score). Moreover, to increase the effectiveness of bankruptcy prediction, the external factors also can be added to the BPCM model [163], [164] – for example, some macroeconomic indicators like GDP, Inflation Rate, Market Indexes, etc.

Looking ahead, the ML Engine doesn't require many input variables, as excessive data could overfit it. The tools described in the following sections will help reduce the ratios to a reasonable amount.

Table 1: Financial Ratios used as Input Data for Bankruptcy Prediction Computational Model

Code	Category Name	Ratio Name
R1	Productivity	Return on Shareholders Funds, Last avail. yr
R2		Return on Capital Employed, Last avail. yr
R3		Profit Margin, Last avail. yr
R4		Gross margin (%), Last avail. yr
R5		Net Assets Turnover (x), Last avail. yr
R6	Liquidity and solvency	Current ratio (x), Last avail. yr
R7		Liquidity ratio (x), Last avail. yr
R8	Business activity	Creditors Payment (x), Last avail. yr
R9		Debtors Turnover (x), Last avail. yr
R10		Stock Turnover (x), Last avail. yr
R11	Financial sustainability (stability)	Gearing (%), Last avail. yr
R12		Interest Cover (x), Last avail. yr
R13		Cash Flow Coverage Ratio (x), Last avail. yr
R14		Current Liability Coverage Ratio (x), Last avail. yr

4.2 Developing Ontology of Bankruptcy Prediction

For this use case, I developed an *Ontology of Bankruptcy Prediction (OBP)* , which is an ontology containing information about financial analysis rules and regulations used by current analytical experts in this field. What is more critical, OBP Ontology includes information about financial ratios used in bankruptcy prediction, how these ratios are interconnected, and what kind of financial processes take place in a company depending on changes in these ratios' values.

[22]https://www.ifrs.org

[23]http://www.legislation.gov.uk/ukpga/2006/46/contents

The composite Ontology of Bankruptcy Prediction content is based on the IFRS Standards and the UK Companies Act 2006, which sets the principles for preparing and analysing financial statements.

To create the OBP Ontology, I initially define an informal conceptual map shown in Fig. 8 and then a formal physical representation – an OWL file (see Appendix H).

The upper part (yellow) of the OBP Ontology represents the structure of the Elements of a Company's Financial Statements (e.g. Turnover, Operating Cash Flow, Current Liabilities, etc.), which are taken from the company's financial statements (records) – Cash Flow Statement, Profit & Loss Account, and Balance Sheet.

The lower part (green) contains knowledge about the Ratios that characterise the financial conditions of a company. Currently, the OBP ontology includes 14 indicators most often used in bankruptcy prediction models: Return on Capital Employed, Liquidity Ratio, Cash Flow Coverage, Gearing, etc. The upper and lower parts of the OBP Ontology are interconnected by various types of non-hierarchical relationships that determine the nature of the dependencies of Financial Statements' Elements and Ratios. For example, the *directlyRelatedTo* relationship shows the connection of the entities, which are decreasing/increasing in equal proportions.

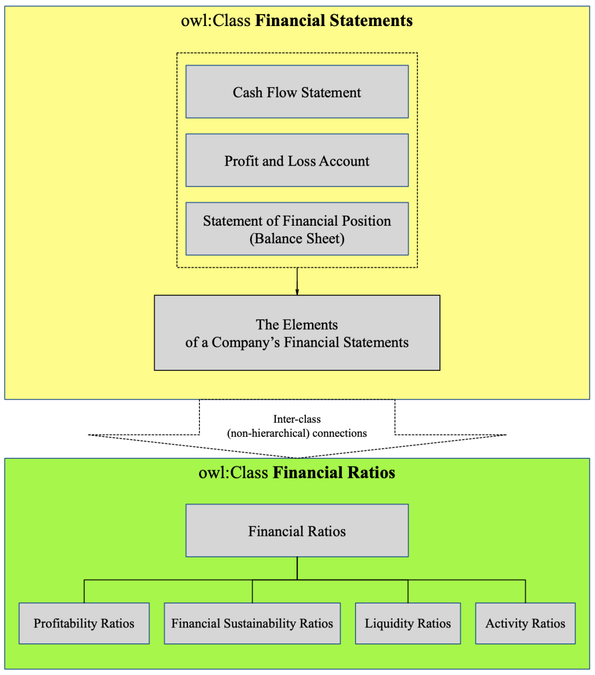

owl:Class **Financial Statements**
Cash Flow Statement
Profit and Loss Account
Statement of Financial Position (Balance Sheet)
The Elements of a Company's Financial Statements
Inter-class (non-hierarchical) connections
owl:Class **Financial Ratios**
Financial Ratios
Profitability Ratios
Financial Sustainability Ratios
Liquidity Ratios
Activity Ratios

Figure 8: OBP Ontology Framework

The detailed Ontology of the Bankruptcy Prediction diagram is given in Appendix B.

Noting that the ontology structure is a graph, I can illustrate these types of the OBP Ontology interconnections by an example of one of its paths – the components of the *Gearing* Ratio (see Fig. 9).

Gearing is a financial indicator, one of 14 input ratios for BPCM model, responsible for the balance of borrowed and owned funds. High Gearing may potentially indicate extreme credit risk.

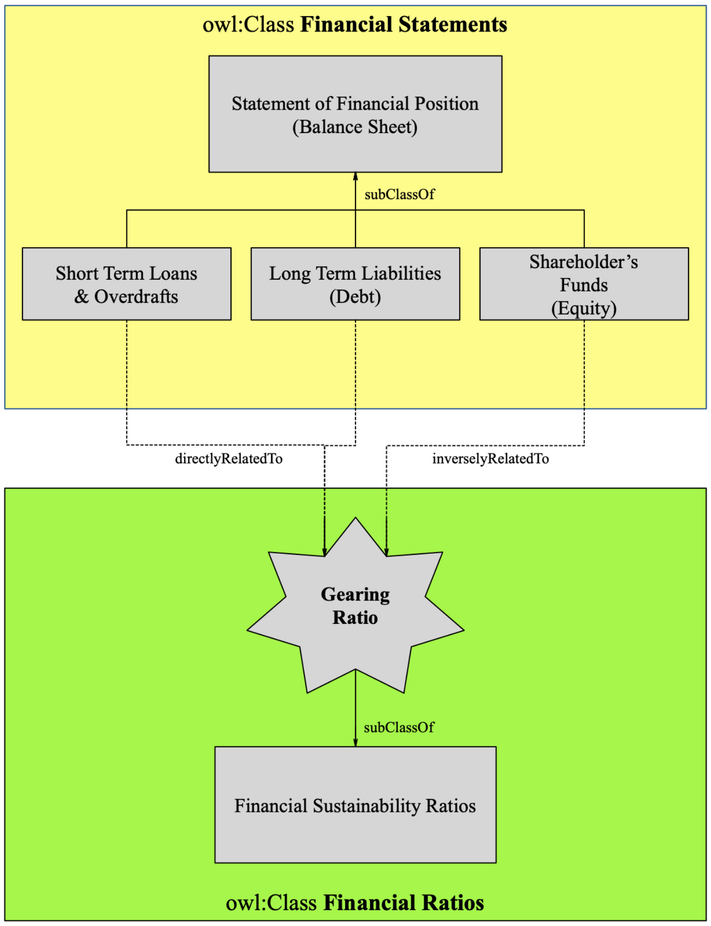

owl:Class **Financial Statements**
Statement of Financial Position (Balance Sheet)
subClassOf
Short Term Loans & Overdrafts
Long Term Liabilities (Debt)
Shareholder's Funds (Equity)
directlyRelatedTo
inverselyRelatedTo
Gearing Ratio
subClassOf
Financial Sustainability Ratios
owl:Class **Financial Ratios**

Figure 9: Ontological Path of Gearing Ratio Components

This Ratio is calculated using three Elements of Financial Statements – Short Term Loans & Overdrafts plus Long-Term Liabilities, divided by Shareholders' Funds. All of them are components of a company's Balance Sheet.

Thus, the changes in the "Short Term Loans & Overdrafts" and "Long-Term Liabilities" indicators are directly proportional to the changes in the Gearing Ratio. At the same time, the changes in the "Shareholders' Funds" indicator affect the Gearing Ratio inversely. This is shown in Fig. 9by means of the *directlyRelatedTo* and the *inverseryRelatedTo* relationships.

Once the structure is created, the next step is to build an OBP Ontology in a software environment and generate an OWL file. Here I utilise Protégé (see Section 3.4).

At the first stages of the research, I had a problem transferring the composite OBP Ontology file to Neo4j Environment since the latter does not recognise data attributes converting all communications other than hierarchical into separate nodes. Thus, I had to divide the OBP Ontology into two parts: data from accounting documents and financial ratios. These two parts are hierarchical class taxonomies with one type of relationship between components – a SubClass of (SCO). I described the process of transferring these split ontologies in my published paper [165].

Lately, I have found a more elegant way to manage and transfer the whole ontology with all the metadata using the Python environment. It is discussed below.

The Full Ontology hierarchy developed in the Protege environment is shown in Fig.10.

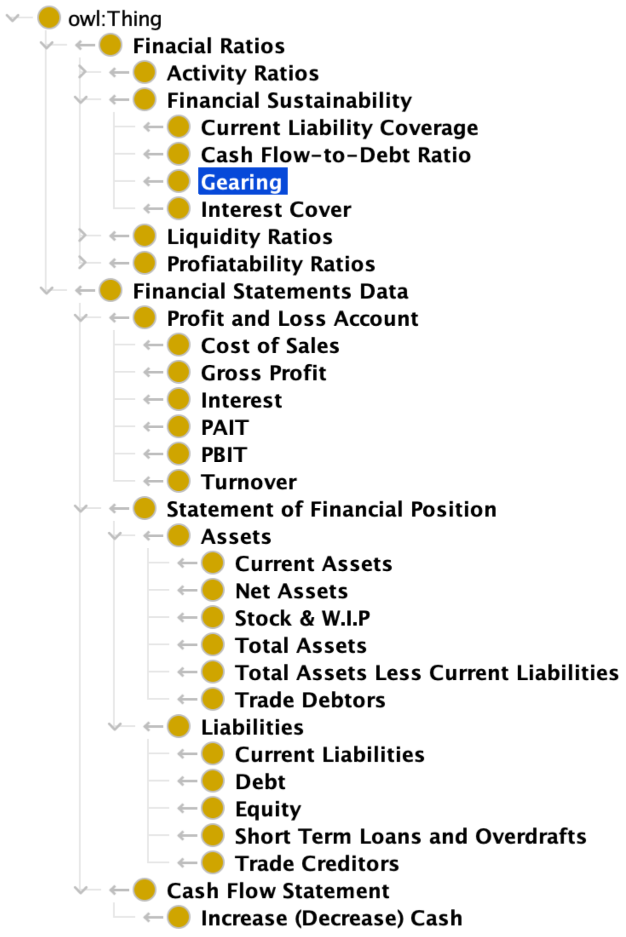


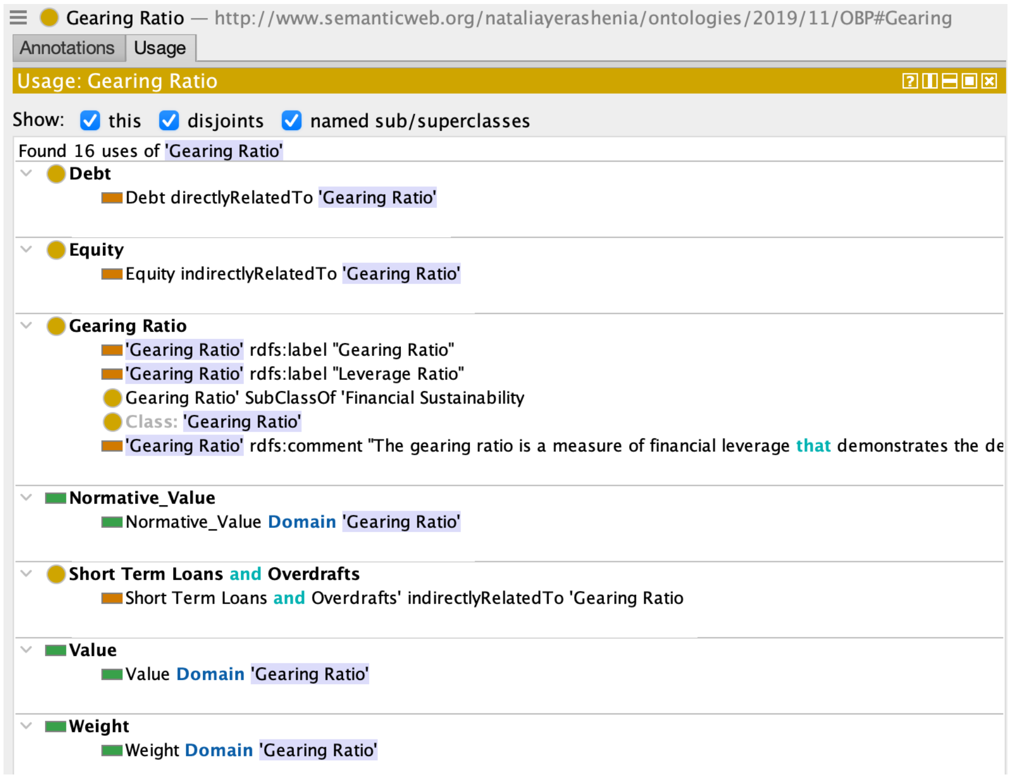
Figure 10: Protégé Environment: Full Ontology of Bankruptcy Prediction

The Ontology contains two main classes: Financial Ratios and Financial Statements data. Except for the SCO relationships, the OBP Ontology includes two types of non-hierarchical connections: *directlyRelatedTo* and *indirectlyRelatedTo*. These connections are used to connect the entities from the two main classes.

It should be noted that at this stage, the OBP Ontology has a simplified structure and contains 14 ratios only. However, in the future, it can be expanded by adding more data units, depending on the users' needs.

In creating an ontology, I specify data attributes, "empty containers" for information about the indicators of a particular company (such as the Value of the Financial Statements' Elements in a given year, the Value of the Elements in the past few years, the industry normative value, etc.). These values will be subsequently filled with data in a Graph DB (see Fig.3.4).

In Fig.11 one may see the 'Usage' tab of Gearing Ratio in Protege. It shows that the entity contains three data attributes: *Weight* (which is the ratio importance coefficient), *Value* (the actual value of the Ratio) and *Normative_Value* (it can be filled if the company has a unique threshold value for a particular Ratio). The rest of the Ratios contain the same data attributes. Also, this tab shows the relations of the Ratio with other entities in the Ontology. Moreover, every entity includes information about synonyms and term definitions.


Gearing Ratio — http://www.semanticweb.org/nataliayerashenia/ontologies/2019/11/OBP#Gearing

Annotations | Usage

Usage: Gearing Ratio

Show: ✓ this ✓ disjoints ✓ named sub/superclasses

Found 16 uses of 'Gearing Ratio'

Debt
- Debt directlyRelatedTo 'Gearing Ratio'

Equity
- Equity indirectlyRelatedTo 'Gearing Ratio'

Gearing Ratio
- 'Gearing Ratio' rdfs:label "Gearing Ratio"
- 'Gearing Ratio' rdfs:label "Leverage Ratio"
- Gearing Ratio' SubClassOf 'Financial Sustainability
- Class: 'Gearing Ratio'
- 'Gearing Ratio' rdfs:comment "The gearing ratio is a measure of financial leverage that demonstrates the de

Normative_Value
- Normative_Value Domain 'Gearing Ratio'

Short Term Loans and Overdrafts
- Short Term Loans and Overdrafts' indirectlyRelatedTo 'Gearing Ratio

Value
- Value Domain 'Gearing Ratio'

Weight
- Weight Domain 'Gearing Ratio'

Figure 11: Protégé Environment: Gearing Usages in OBP Ontology

An LSTM file containing the full version of OBP Ontology – *OBP_Ontology_v.2.1.owl*[24] is used to create the skeleton of the BPCM Model Graph DB in Neo4j.

Besides, the Protégé the OWL file can be easily managed and read in the Python environment using the *owlready2* library. I will use this when transferring the file to the Neo4j environment.

4.3 Developing Semantic Graph Database

The first step in creating a Neo4j Graph DB for a company is to transfer the OBP Ontology file made in Protégé into Neo4j.

The way of importing the Ontology to Neo4j using Cypher[25] only is described in my published paper [165].

[24]From now on, I will use the name of the files only to maintain the flow of the text. However, the naming of the files reflects their functionality. See the full list of files in Appendix H.

[25]`https://neo4j.com/docs/labs/nsmntx/current/importing-ontologies/`

However, in the updated version of the system for these purposes, the Python code is used. So now, the process of transferring is fully automated.

A separate file is created in the Python environment, where the code for transportation will be placed – see the file *import_to_neo4j_uc1.py* attached.

First, I should create and open an empty graph called *'BPCM'* (the name doesn't matter, though) in the Neo4j app window. Then define the LAN connection parameters to connect to the Graph DB in Python.

Now I need to import the OBP file itself (the owlready2 library will help us with this). After that, I can proceed directly to extracting information from the OBP ontology and using it as a base for building a graph.

```
if __name__ == "__main__":
greeter = neo4j_import("bolt://localhost:7687", "neo4j", "000000")
onto_path.append(".")
onto = get_ontology("OBP_Ontology_v.2.1.owl")
onto.load()
greeter.reset_graphs()
greeter.add_nodes_in_neo4j(onto)
greeter.add_annotation_edges_in_neo4j(onto)
```

Nodes and Relationships are created in Python using the Cypher queries.

All elements of the ontology are transferred as graph nodes. In this case, it is necessary to get rid of all prefixes in the names of entities, which are automatically added to the LSTM file. Then hierarchical relationships between entities are added, and then the relationships that are hidden in the annotations of the ontology (non-hierarchical). Thus, an *Ontology template graph* is formed in the environment (see Definition 3.3).

To convert an Ontology template graph to an *Ontology full graph* (see Definition 3.2), a graph that is already filled with data, I need to use the file with a company's data taken from an external source (for example, the traditional database which the company currently uses to keep their financial records). So the first step is to export the file itself to the Python environment and then correlate the names of the created nodes with the names of the columns in the file.

Each Element of a Company's Financial Statements should contain a node attribute called: "Value", which is supposed to reflect the financial data of a particular company I analyse at the moment. It can be in a form of a CSV file. As an example, I use the 2019 data of random Company A (see Section 4.5) – *Company_A_Fin_Indicators.csv*. In my case, a company dataset is provided by Fame Bureau Van Dijk database[26].

Loading a Company's dataset and identifying the column's names:

```
company_fin_indicators = pd.read_csv("Company_A_Fin_Indicators.csv").values[0]
```

[26] https://fame.bvdinfo.com/

```
company_fin_indicators_columns =
pd.read_csv("Company_A_Fin_Indicators.csv").columns.tolist()
company_fin_indicators_dict = {
    "Increase(Decrease) Cash & Equiv.\nth GBP Last avail. yr":
    "Operating_Cash_Flow",
    ...
    "Shareholders Funds\nth GBP Last avail. yr": "Shareholders'_Funds_(Equity)"}
```

Matching the dataset columns' names with the entity's names in Ontology:

```
    match_nodes_query = "MATCH "
IRI_prefix_2 = "OBP_Ontology_v.2.1."
for c in onto.classes():
    cname_formatted = c.name\
        .replace(IRI_prefix_2, "")\
        .replace(".", "_")\
        ...
        .replace("&", "_")
    match_nodes_query += "({cname_formatted}:Class {{name: '{cname}'}}), " \
    .format(cname_formatted=cname_formatted,
    cname=c.name.replace("'", "\\'"))
set_values_query = "SET "
i = 0
for col_name in company_fin_indicators_columns:
    matching_node_name = company_fin_indicators_dict[col_name]
    matching_node_name = matching_node_name\
        .replace(".", "_")\
        ...
        .replace("&", "_")
    set_values_query += f"{matching_node_name}.Value = {company_fin_indicators[i]}, "
    i += 1

match_nodes_query = match_nodes_query[:-2]
set_values_query = set_values_query[:-2]

with greeter.driver.session() as session:
    session.run(match_nodes_query + " " + set_values_query)
```

The next step is to calculate and fill the "Value" attributes of the Ratios by using built-in math formulae. For example, the formula of Gearing Ratio in Cypher can be shown as:

```
MATCH (gear:Resource {name: "Gearing"})
SET gear.value = (-(stlao.value + ll.value) / sf.value) * 100
RETURN gear.value;
```

It should be noted that Ratios (given in Table 1) formulas were taken from the Fame database as well to compare the obtained results with the corresponding ready-made data subsequently.

To save the necessary data from the Graph DB for transferring to other model components, a separate new CSV file called *Ratios_Export.csv.* is generated.

After the first iteration of the model, the system generates a CSV file with the weights of the model input indicators called *weights_importance_uc1.csv* (see below how this file is created). After exporting and processing this file I form the next "layer" of the Graph DB - *Ontology full graph with weights* (see Definition 3.1).

```
weights = pd.read_csv("weights_importance_uc1.csv", header=0).values
weights = weights.T
weights_columns = np.array(weights[0]).tolist()
weights = weights[1]
set_weights_query = "SET "
for c in onto.classes():
    if str(c.name) in weights_columns:
        index_for_weight = weights_columns.index(c.name)
        set_weights_query += f"{c.name}.Weight = {weights[index_for_weight]}, "
set_weights_query = set_weights_query[:-2]
```

The whole Ontology of the Bankruptcy Prediction graph with weights, which is built in the Neo4j environment, is given in Appendix C.

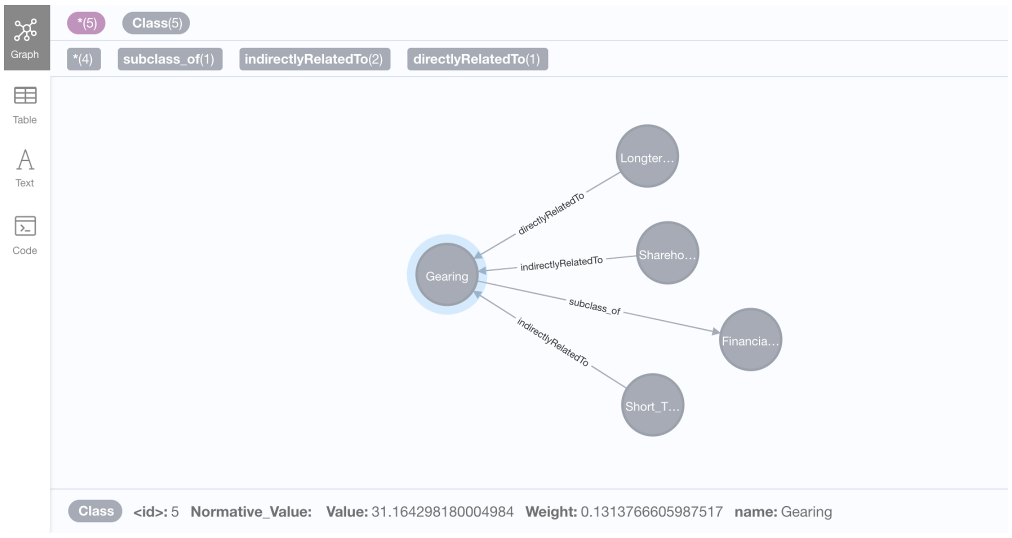
Next, based on the weights obtained, I can proceed with the *Feature Selection.* I can manage this Python environment in my case I developed *cypher_queries_uc1.py*) file.

Then, through specific queries from several indicators in one category, one with the maximum weight is selected. At the same time, Neo4j should display a graph containing only the indicators remaining after the selection. Finally, the data of this graph is exported as a CSV file for transporting it to the next component of the system.

According to Table 1 the Ratios in BPCM Model are divided into 4 categories. For example, the feature selection Cypher query for the Financial Sustainability category should be present as:

```
Financial_Sustainability_node = ''
...
with neo4j_db.driver.session() as session:
...
   Financial_Sustainability_node = session.run("""MATCH (n1:Class {name:
    "Financial_Sustainability"})<-[r1:subclass_of]-(scn1:Class)
     WITH max(scn1.Weight) AS maximum
     MATCH (n2:Class {name: "Financial_Sustainability"})<-[r2:subclass_of]
     (scn2:Class)
     WHERE scn2.Weight = maximum
     RETURN scn2""").data()
...
fs_node_name = Financial_Sustainability_node[0]["scn2"]["name"]
max_weighted_features = [
     ...,
     fs_node_name,
     ...,
     ... ]
```

The result of this query is shown in Fig.12.



Class <id>: 5 Normative_Value: Value: 31.164298180004984 Weight: 0.1313766605987517 name: Gearing

Figure 12: Neo4j Environment: Returning the Ratio with the maximum weight in Financial Sustainability category (2019 dataset example)

Then the structured and selected Features (in my case, financial ratios) are transferred to the Python environment as a CSV file called *Input_Data_with_Neo4j_Feature_Selection.csv*, which is served as input data for the NN (starting from the second model's cycle).

4.4 Developing Machine Learning Engine: Classical Neural Network

After the data have been structured, selected and saved in a convenient format by means of OBP Ontology and the Neo4j Graph DB, I can proceed directly to the data analysis – evaluation of a company's bankruptcy level.

Determining the financial conditions of companies is a solution of the *classification problem*, i.e. assigning of available data samples to specific classes [166].

Classification using Neural Networks. When dealing with classification problems, assigning the available static units (company information) to specific classes is necessary. Various ways of presenting data are possible, with the most common approach being to represent a statistical unit as a vector. The components of this vector represent different characteristics of the unit that influence the decision of which class the company should be assigned to. Thus, based on information about a company, it is necessary to determine its class. A classifier assigns an object to one of the classes based on a particular partition of the N-dimensional space, known as the input space, where the dimension of this space corresponds to the number of vector components.

The first step is to determine the level of complexity of the system. In real problems, the number of samples is often limited, making it challenging to determine the complexity level. Three main levels of difficulty can be distinguished. The first level is linear separability, where straight lines or hyperplanes can separate classes if the input space has a dimension higher than two. The second level is non-linear separability, where lines or hyperplanes cannot separate classes but can be separated using more complex divisions. The third level is probabilistic separability, where classes intersect, and probabilistic separability methods are applied.

Two sets are considered linearly separable if at least one line (or hyperplane in higher dimensions) exists on the plane such that all points of each set lie on one side of the line. This definition can be extended to spaces of higher dimensions. If two sets of vectors in a space cannot be separated by a straight line or plane, they are called linearly inseparable [167].

Ideally, after pre-processing, the goal is to achieve a linearly separable problem, as it simplifies the construction of the classifier. However, in real problems, the number of samples is limited, making it challenging to achieve linear separability.

When applying neural networks in practice, several challenges arise. Firstly, it is still being determined in advance what complexity (size) the network may require to implement a mapping accurately. It is possible that the complexity level may be excessively high, requiring a complex network architecture. According to Minsky and Papert [168], simple single-layer neural networks can only solve linearly separable problems. This limitation can be overcome by using multilayer neural networks. By employing a network with one hidden layer, the input sample vector is transformed by the hidden layer into a new space, which may have a different dimension. The hyperplanes corresponding to the neurons in the output layer then divide this space into classes. Therefore, the network recognises not only the features of the data but also the features formed by the hidden layer.

To build a classifier, it is necessary to determine which parameters influence the decision of which class a sample belongs to. This can give rise to two problems. Firstly, if the number of parameters is small, there may be situations where the same set of source data corresponds to examples in different classes. In such cases, it is impossible to train a neural network, and the system will not work correctly as it cannot find a minimum corresponding to such a set of source data. Therefore, the source data must be consistent. To address this problem, it is necessary to increase the dimension of the feature space, i.e., the number of components in the input vector corresponding to the sample. However, increasing the dimension of the feature space may lead to a situation where the number of examples becomes insufficient for training the network. Instead of generalising, the network may remember the examples from the training set and need help functioning correctly. Thus, when determining the characteristics, a compromise must be found regarding their number. This dilemma can be resolved through feature selection.

Next, the method of representing the input data for the neural network, i.e., the normalisation method, must be defined. Normalisation is necessary because neural networks typically operate with data represented by numbers from 0 to 1, while the source data may have arbitrary ranges or even be non-numeric. Several methods are possible for normalisation, such as simple linear transformation into the required range or multidimensional analysis of parameters and non-linear normalisation based on the influence of parameters on each other.

When dealing with a classification problem involving two classes (bankrupt, non-bankrupt), a network with one neuron in the output layer can be used, which takes values of 0 or 1 depending on the class to which the sample belongs. However, presenting the data becomes more challenging when there are multiple classes.

The cascade correlation method is one approach where the network architecture is adjusted to minimise the error after each iteration [169]. This approach starts with an over-complete network and then removes nodes and connections that have little effect on the solution. However, a rule must be followed: the number of examples in the training set must be higher than the number of adjustable weights. Otherwise, instead of generalising, the network will remember the data and lose its ability to classify, resulting in an undefined outcome for examples not included in the training set.

Based on information from various sources [2, 13, 170, 171], the process for constructing a classifier based on neural networks includes the following steps:

1. Data collection:
   - Compile a database of examples specific to the task.
   - Divide the entire data set into training and testing (it is possible to include a validation set).
2. Data pre-processing:
   - Select a specific feature system for the task and convert the data accordingly to feed into the network (normalisation, standardisation, etc.).
3. Network design, training, and quality assessment:
   - Choose a suitable environment for building the neural network.
   - Determine the network topology, including the number of layers and neurons in each layer.
   - Select an activation function for the neurons and a learning algorithm for the network.
   - Evaluate the network quality based on the validation set or other criteria and optimise the architecture (reducing weights, feature space reduction).

- Choose network settings that provide the best generalisation ability and evaluate the quality of the network on the test set.

4. Usage and diagnostics:

    - Determine the degree of influence of various factors on the decision using a heuristic approach.
    - Ensure that the network achieves the required classification accuracy (low number of misclassified examples).
    - If necessary, return to step 2 by changing the data representation or modifying the database.
    - Use the network in practice to solve the problem.

Now, let's proceed with implementing these steps in our specific scenario.

Collecting the Data. For supervised network training, it is necessary to prepare a set of training data: examples of input data and their corresponding outputs.

The financial data of UK middle-size companies for 2017-2021[27], including the 14 financial ratios (see Table 1) was chosen as the dataset for BPCM. The complete dataset[28] containing *451 observations* was taken from *Fame Bureau Van Dijk* database and saved as five separate CSV files, one for each financial year (see Appendix H). For example, the file containing 2019 year data is named *Training_Output_2019_87.csv*

According to the independent auditors' reports, provided at Companies House website[29] the bankruptcy level of the companies was evaluated. Hence the companies were decided into two groups: bankrupts and non-bankrupts, see Table 2.

Table 2: The number of bankrupt and non-bankrupt observations in the BPCM complete dataset

Financial Year	2017	2018	2019	2020	2021
Bankrupts	23	39	37	36	47
Non-bankkrupts	48	47	50	58	66
Total number of observations	71	86	87	94	113

The testing dataset is randomly selected and is set to comprise 10% of the complete dataset. The rest were used as training data for the model. The class of each observation (0 or 1) is included in the training datasets (CSV files) – column 'Class'.

[27]This time period includes the period of COVID-19 pandemic, which, to some extent, brings disruption to data analysis

[28]The *complete dataset* comprises training and testing data, i.e., all the data used during the model development process, from training through testing. A training dataset is used to train or build the model. It provides the raw material for learning the predictive patterns and features. Conversely, a testing dataset is used to assess the performance of the trained model. It tests the model's capability to generalise to new, unseen data. The model must perform well on this dataset, as it reflects how the model will perform in the real world.

[29]`https://beta.companieshouse.gov.uk`

Pre-processing of Data. As it was stated before, a pre-processing of data has a major impact on data analysis quality [13]. This part has already been done by means of the Semantic DS. See Sections 4.2 and 4.3.

The BPCM Neural Network Design. To build a Neural Network for BPCM model, I use a Python environment. Typically, the TensorFlow or Keras packages are used to create a neural network in Python – the libraries suitable for a wide range of machine learning techniques. However, my task is to create the network from scratch, as in the future, the network will have to be tuned and adapted to the model's further updates. To implement a neuron network, I use *NumPy* – a powerful Python library that handles maths operations.

A single-layer perceptron or a feed-forward neural network (NN) is a common approach to address the classification problem [172], [102]. It consists of an input layer, a hidden layer with weighted connections and an activation function like the sigmoid, and an output layer. The input layer receives data, which is then processed by the hidden layer neurons using weights and an activation function. The output layer produces the final predictions. Training involves adjusting the weights using backpropagation, minimising the error between predicted and desired outputs. For a more detailed survey of Classical Neural Networks, see Section 2.6.2.

In NN, an artificial neuron receives signals through several input channels. Each input signal passes through a connection (synapse) having a specific intensity (weight) [111]. The current state of a neuron is defined as the weighted sum of its inputs:

$$\sum_{i=1}^{N} w_i \times R_j, 1 \leq j \leq N,$$

where w_i are synaptic weights of input signal j and Rj – input signals (ratios), N is a space dimension of the input signals.

To produce an output, the following formula can be used:

$$\sum_{i=1}^{N} F(w_i \times R_j) + b_j, 1 \leq j \leq N,$$

where $F(w_i \times R_j)$ is an activation function, and b_j – bias vector for the neuron j.

For the analysis of financial data, a Sigmoid (logistic) activation function can be used [101]. The *Sigmoid function* is commonly used in classification neural networks due to its ability to map inputs to a range between 0 and 1, making it suitable for binary classification tasks [173], [174]. Its non-linearity allows neural networks to capture complex relationships and learn non-linear decision boundaries. The smoothness and differentiability of the Sigmoid function enable efficient gradient computations during training. The Sigmoid function also provides a probabilistic interpretation, assigning probabilities to classes based on its outputs. It stabilises outputs and is widely used and familiar in machine learning. While alternative activation functions have gained popularity, the Sigmoid function remains a viable choice, especially for binary classification and probability interpretation.

Mathematically, the Sigmoid function can be presented as:

$$F = \frac{1}{1 + e^{w_i \times R_j + b_j}}$$

In Python, the Sigmoid function looks as follows:

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

In Fig. 13, the architecture of the Neural Network built for the BPCM model is provided. In this Figure *R_1, R_2, R_3, .., R_14* – are the financial ratios of a considered company, and O – is the output of the NN. In my case, there are two classes of output: [0.] – a company with a stable financial position, [1.] – a bankrupt company.

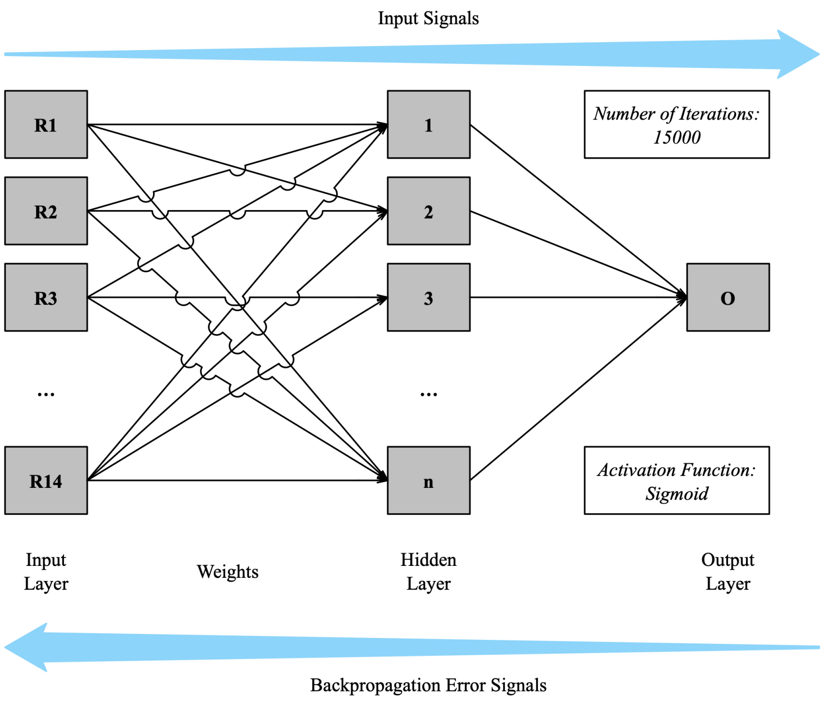

Input Signals
R1
R2
R3
...
R14
1
2
3
...
n
O
Number of Iterations: 15000
Activation Function: Sigmoid
Input Layer
Weights
Hidden Layer
Output Layer
Backpropagation Error Signals

Figure 13: The Architecture of the BPCM Neural Network

It can be seen that the bpcm NN I created represents a classical neural network with one layer, backpropagation, and a sigmoid activation function. It consists of an input layer, a hidden layer with sigmoid activation, and an output layer. The input layer receives the input data, which is then connected to the neurons in the hidden layer. Each neuron in the hidden layer applies a weighted sum of the inputs and passes it through the sigmoid activation function. The output layer produces the final predictions based on the weighted sums from the hidden layer. During training, the network uses backpropagation to adjust weights and minimise errors between predicted and desired outputs. This process is done by propagating the error backwards from the output layer to the hidden layer, and updating the weights using gradient descent. The sigmoid function introduces non-linearity and enables the network to capture complex relationships.

After defining the bpcm ML Engine architecture, the next step is to train the NN. In our case it will consists of 150 iterations.

As an input source, a file with examined company ratios is used, which was exported in CSV format from Neo4j (*Ratios_Export_NN.csv*, see Appendix H).

The "DictReader" class from the CSV module in the Python standard library reads through the row containing the ratios in the CSV file. Individual Ratio values are singled out by specifying the column name the Ratio was under, which is then stored in program variables. The variables are then put into an array passed as input for the Sigmoid function to perform computations. As a result, the neural network program written in Python prints a value of [0.] or [1.].

BPCM Neural Network Evaluation. To make sure that the developed NN is working accurately and can be used for financial company analysis purposes, I should test it.

I ran the Neural Network code (see *BPCM_NN_testing.py* file) on the testing dataset which comprises data from 2017 to 2021 of 45 companies (10% of complete dataset).

To assess the results, I used Classification Report and Confusion Matrix from *SKLearn.metrics*[30] library.

The results of the BPCM NN evaluation are presented in Table 3.

Table 3: The accuracy results for BPCM Neural Network

Financial Year	2017	2018	2019	2020	2021
Training Dataset Size, companies	7	9	10	9	10
Incorrectly Identified Bankrupt Companies	1	0	1	0	0
Incorrectly Identified Non-Bankrupt Companies	0	1	0	0	0
Total accuracy	0.86	0.94	0.95	1	1

Once NN is passed the testing, the network is ready for use.

Calculating the Weights for the FeedbackLoop. After the file with written NN code for BPCM Model (*BPCM_NN_v.2.0.py*) is executed the *feature importance analysis* of the input data should be made. For this purposes the special Python code is used – *feature_importance_uc1.py*.

Loading input data:

```
df = pd.read_csv('Training_Data_2019_88.csv')
df.fillna(0, inplace=True)
```

[30] https://scikit-learn.org/stable/modules/model_evaluation.html

```
training_data = df.drop('Class',axis=1)
training_output = df['Class']
training_data_cols = np.array(df.drop('Class',axis=1).columns.values.tolist())
```

Fitting an Extra Trees model to the data:

```
model = ExtraTreesClassifier()
model.fit(training_data, training_output)
```

Displaying the relative importance of each attribute

```
print(model.feature_importances_)
weights = np.array(model.feature_importances_)
training_data_cols_matrix = np.expand_dims(training_data_cols, axis=1)
weights = np.expand_dims(weights, axis=1)
```

Table output:

```
table = np.concatenate([training_data_cols_matrix, weights], axis=1)
table = pd.DataFrame(table)
table.columns = ['Attribute', 'Weights']
table.to_csv('weights_importance_uc1.csv', index=False)
```

As one may see, the code uses the *Random Forest Feature Importance* method to evaluate the importance of the input ratios' weights. Finally, the result is saved as a CSV file (*weights_importance_uc1.csv*), which is further transferred to the Graph DB (as a part of the feedback loop).

BPCM Neural Network Evaluation after the model's first cycle. To assess the Neural Network's accuracy after the model's first cycle, I used the same algorithm as described in 'BPCM Neural Network Evaluation' paragraph above. However, in the testing datasets (2017 – 2021), I left only those four ratios identified as most valuable after the feature selection for each year.

The results of the BPCM NN evaluation are presented in Table 4.

Table 4: The accuracy results for BPCM Neural Network after the first model's cycle (after the feature selection)

Financial Year	2017	2018	2019	2020	2021
Training Dataset Size, companies	7	9	10	9	10
Incorrectly Identified Bankrupt Companies	0	0	0	0	0
Incorrectly Identified Non-Bankrupt Companies	0	0	0	0	0
Total accuracy	1	1	1	1	1

Thus, the results were improved if compared with Table 3 and reached the accuracy of 100% through all five years. This supports the efficiency of the entire BPCM model.

Comparison BPCM Neural Network and Classic Logistic Regression Model Results. To gain a more comprehensive understanding of the efficacy of the BPCM Neural Network, it is essential to compare it with another robust model. In this regard, I utilised a simple Logistic Regression model (Using the SKLearn LogisticRegression package in Python), trained with identical datasets used for the BPCM Neural Network

The result is presented in *BPCM_LogReg_testing.py* file. The testing revealed that the Logistic Regression model exhibited an overall accuracy of 91.67%, which is commendable given its simplicity. However, when we juxtapose this against the results of the BPCM Neural Network, some insightful observations emerge.

Firstly, with an accuracy score ranging from 0.86 to 1 (depending on the financial year), the BPCM Neural Network demonstrated superior or comparable performance to the Logistic Regression model. This disparity became even more pronounced after the model's first cycle, where the BPCM Neural Network achieved an unblemished accuracy score of 1 across all financial years.

Secondly, as reflected in the confusion matrix and classification reports (see *BPCM_NN_testing.py*), the BPCM Neural Network appeared to be more proficient at correctly identifying bankrupt and non-bankrupt companies. In comparison, while delivering good results, the Logistic Regression model has a marginally lower precision, recall, and F1-score.

Hence, while the Logistic Regression model exhibits substantial predictive power, the BPCM Neural Network, especially after its first cycle, outperforms it, underscoring the validity and robustness of the BPCM Neural Network for predicting bankruptcy.

These findings lend credence to using sophisticated machine learning models like the BPCM Neural Network in the financial sector, particularly for complex tasks such as bankruptcy prediction.

BPCM Neural Network Cross-Validation. *Cross-validation*, such as 10-fold cross-validation, is a technique used to assess the performance of a neural network or any other machine learning model and to safeguard against overfitting [175]. Overfitting occurs when a model learns the training data too well, so it performs poorly on unseen data. In the context of 10-fold cross-validation, the original dataset is randomly partitioned into ten equal-sized subsets or "folds".

In each round of validation, one fold is retained as the testing data, and the remaining nine folds are used as training data. This process is repeated ten times, with each fold used exactly once as the testing data. The ten results from the folds can then be averaged for a single estimation. The advantage of this method is that it matters less how the data gets divided because each data point gets to be in a test set precisely once and gets to be in a training set nine times. This provides a more reliable measure of the model's predictive power, as it is averaged across multiple test sets.

10-fold cross-validation was performed using the following SKLearn library for Python

```
# Perform 10-fold cross-validation
scores = cross_val_score(mlp, X_train, y_train, cv=10)


# Output the mean cross-validation score
```

```
print("10-Fold Cross-Validation Score: ", np.mean(scores))
```

For more details, see *BPCM_NN_cross-validation_testing.py* file.

As a result, a 10-fold cross-validation score is 0.9509. It represents the mean accuracy of the model across the ten different folds during cross-validation. Generally, a score above 95% is considered very good in many machine learning tasks, especially if the dataset is balanced.

It's important to note that while a high cross-validation score is a positive sign, it does not guarantee the model will perform as well on unseen data. Remember that the key goal in machine learning is generalisation; that is, the model's ability to predict accurately new, unseen data. High training and validation scores are only useful to the extent that they indicate the model will also score well on unseen data.

4.5 Using the Developed Bankruptcy Prediction Computational Model to Analyse a Bankruptcy Level of a Real Company

Collecting a Company's Data. To explore the approach in a real-life environment and give a better understanding of the methodology presented, I apply the developed software solution of the BPCM model to analyse the bankruptcy level of Company A, which went bankrupt in 2019.

For ethical reasons, I do not give the real name of the company concerned in the use case; subsequently, I refer to it as Company A.

According to the Companies House website, Company A is a private limited company of medium size with a turnover of around £25,500k.

In 2018 it made a loss of around £1,100k. The independent auditors' report states that the company has significant problems with their financial sustainability, and there is an extremely high risk of going bankrupt. Besides, the auditors concluded that substantial additional funding and changing a management strategy, including ongoing cost controls, are required. Otherwise, there is doubt about the company's ability to continue as a going concern.

The financial data of Company A, including the Elements of Cash Flow Statement, Income statement and Balance Sheet, was taken from Fame Bureau Van Dijk database and saved as a CSV file – *Company_A_Fin_Elements.csv* (see Appendix H).

Under real conditions, a company's data can be taken from the company's database and should be saved as a CSV file.

Approbation. Once the dataset needed for the analysis is collected, it is exported to the template Neo4j Graph DB as the "Values" of the particular nodes – Elements of Financial Statements.

There is no need to create the Graph DB and OBP Ontology from scratch in every new case, as they represent the general framework for companies' data pre-processing.

Using the given values of Financial Statements' Elements, the system automatically calculates the Finical Ratios of Company A. The obtained results match with the calculations provided in Fame, so the system works correctly.

A part of a Graph DB for Company A with calculated Gearing ratio "value" attribute has already been revealed in Fig. 12.

Further, the following files should be imported to the Python environment (also see Appendix H):

- The file with the company's Financial Ratios only can be exported from Neo4j as *Ratios_Export.csv* file. This file is used as Input Data in created Neural Network.

- For the training data, in 2019 *Training_Data_2019_87.csv* file with 14 Financial Ratios values of 87 UK medium-sized companies can be used.

After exporting these two datasets to the NN, I returned the result – [1.], which means that Company A is bankrupt. This conclusion resembles the expert's opinion, which means that the NN was built accurately as well.

After the NN result is received I can use *feature_importance_uc1.py* to evaluate the weights of the 14 ratios. This output file *weights_importance_uc1.csv* can be now added to Neo4j Graph by means of a Feedback loop.

The results of the Random Forest Feature Importance Analysis for Company A are presented in Fig.14.

Then by means of Neo4j queries (*cypher_queries_uc1.py*), the weights are used to identify the four most valuable Ratios (one for each category: Activity Ratios, Financial Sustainability Ratios, Liquidity Ratios, Profitability Ratios). As a result, the model picked the following ratios for Company A:

– Debtors Turnover,

– Gearing,

– Liquidity Ratio,

– Profit Margin.

The usage of only four inputs will prevent the NN from overfitting in future, providing more accurate outcomes.

The file containing the four selected Ratios – *Input_Data_with_Neo4j_Feature_Selection.csv* is returning to the NN as the next iteration input data.

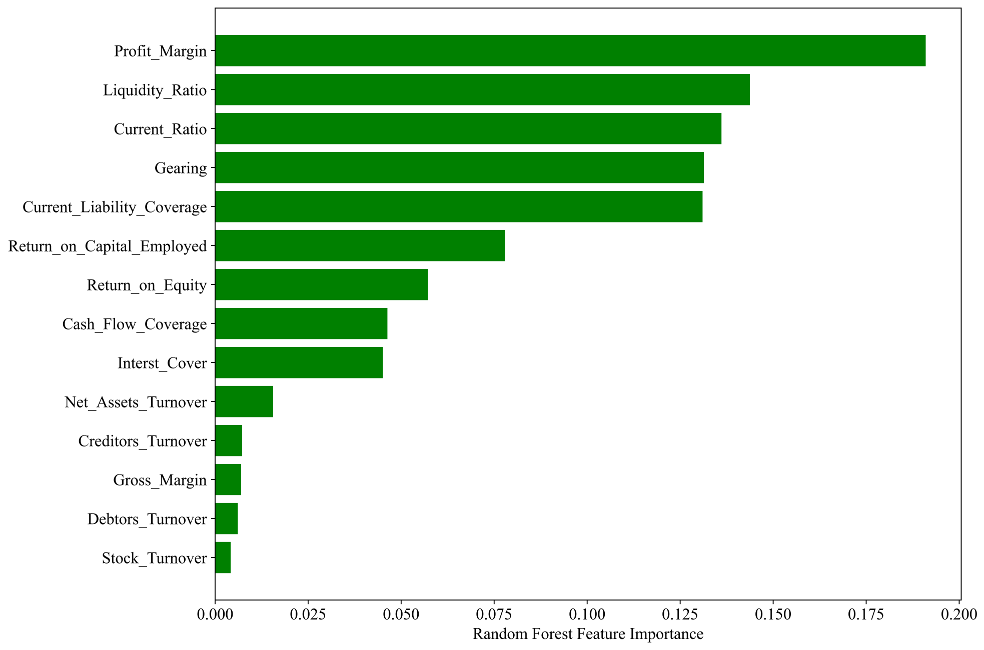


Figure 14: Random Forest Feature Importance Analysis Result for Company A

5 Use Case 2: The Implementation of Market Index Prediction Computational Model

This section describes how to use PCM for the purposes of FTSE100 market index prediction. The use case was described in my IEEE CBI 2022 paper [176].

Market Index Prediction Computational Model (MIPCM) is the special case of PCM, which provides the forecasting prices of the financial market indexes.

The architecture of MIPCM is presented in Fig 7.

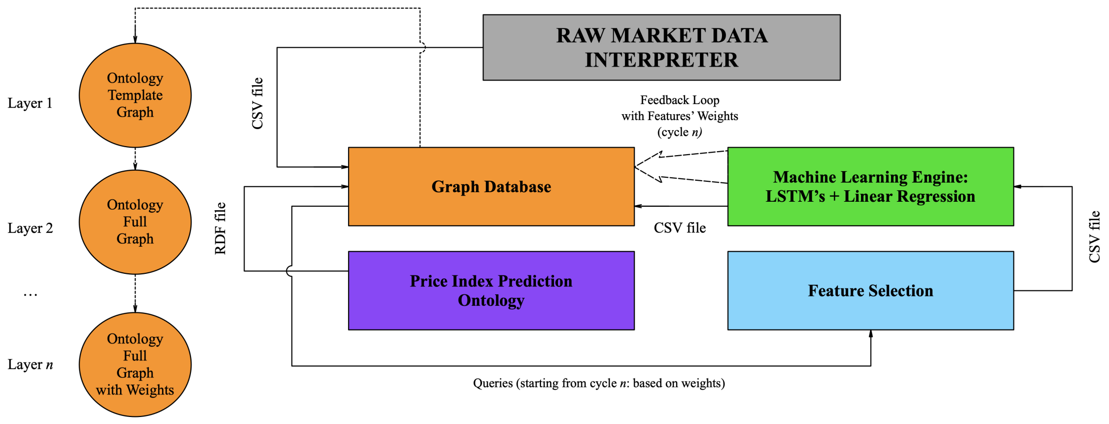

Layer 1
Ontology Template Graph
Layer 2
Ontology Full Graph
...
Layer n
Ontology Full Graph with Weights
CSV file
RDF file
RAW MARKET DATA INTERPRETER
Feedback Loop with Features' Weights (cycle n)
Graph Database
Machine Learning Engine: LSTM's + Linear Regression
CSV file
Price Index Prediction Ontology
Feature Selection
CSV file
Queries (starting from cycle n: based on weights)

Figure 15: BPCM Architecture

5.1 Input Data for Market Index Prediction Computational Model

Macroeconomic statistics have a significant impact on the behaviour of investors in the financial market. Despite the rapid development of algorithmic trading, many financial market participants continue to build their trading strategies based on new data on the current macroeconomic situation. For example, the publication of data on oil stocks can affect the value of quotes for oil futures and other derivatives. Or statistics on the number and volume of new construction can give an idea of the phase of the economic cycle and the state of the economy as a whole, forcing investors to withdraw money from riskier assets and invest in safer companies.

Recently, more articles have appeared in the scientific community devoted to analysing the impact of macroeconomic indicators on financial markets [177], [178], [179], etc.

Standard models for predicting the prices of market indices, such as the NASDAQ, SP500, and FTSE100, use the time series of the previous values of a given index solely [180]. I propose to supplement the input data of MIPCM with the monthly values of some macroeconomic indicators.

FTSE100 (stands for Financial Times Stock Exchange, also known as Footsie) is one of the most popular (including for trading) stock indexes. The FTSE combines data from companies that make up approximately

80% of the London Stock Exchange market capitalisation, and, unlike all other indices, it is not wholly owned by the exchange: its shares are owned by the Financial Times[31].

The FTSE index has been in use since 1984; its original value was 1000 points. In the 2000s, during the Internet boom, the index reached 7103.98. In 2007-2010, the FTSE fell below 3500 due to the financial crisis. Seven years later, in March 2017, the index set its all-time intraday high of 7777.62.

The FTSE100 is comprised of the 100 largest companies listed on the London Stock Exchange; the index data is refreshed and published every 15 seconds.

As the input dataset for MIPCM, I propose to use the average monthly index data from January 1, 1985, to October 1, 2020, that is 430 time-slots (i.e. 1 month for 1 time-slot), and the last 60 slots will be used as test slots for verifying the accuracy of the model prediction.

Basically, the FTSE reflects the state of affairs in the UK stock market; however, some of the companies that make up the index are not British, so the opinion about the FTSE as a purely UK index is not entirely correct.

The composition of the FTSE100 index is determined quarterly; it is unstable and difficult to predict, taking into account the data of companies as of the end of the previous business day. Every quarter, some companies that do not meet specific requirements are dropped from the list, and other companies are put in their place. In this regard, it was decided not to base my research on the performance of the companies that make up the index.

At the same time, most of the companies included in the index are British, political and economic news from the EU also have some impact on the FTSE. However, the main factors still belong to the UK, and news such as decisions on interest rates, GDP, production, and inflation influence the index fluctuations.

Expertly, I have selected 16 macroeconomic indicators that will help improve the predictions of the index. They are presented in Table reftab:mei.

5.2 Developing Ontology of Market Index Prediction

For this use case, I developed an *Ontology of Market Index Prediction (OMIP)* , which is a special ontology containing information about market indexes, their ingredients and calculation rules as well as regulations used by current analytical experts in this field. What is more important for us, OMIP ontology includes the information about macroeconomic indictors which has an impact on Market Index values and how these indicators are interconnected with each other and the market indexes, and what kind of processes normally take place depending on changes in these ratios' values.

Although there are no UK or International standards (as regards OMIP Ontology) which define the relationships between macroeconomic indicators and the market indexes, the OMIP Ontology is based on several experts' opinions.

A formal physical representation of the OMIP Ontology is created in the Protege environment. It is far more complex than OBP ontology (Use Case 1).

At the moment OMIP Ontology contains the following four classes:

[31] `https://markets.ft.com/data/indices/tearsheet/summary?s=ftse:fsi`

Table 5: Macroeconomic Indicators used as Input Data for Market Index Prediction Computational Model

Code	Category Name	Indicator Name
GDP	Economic Output	Gross Domestic Domestic
GNIph		Gross National Disposable Income
EG		Economic Growth
IR	Prices	Retail Price Inflation Index
CPR		Consumer Price Inflation Index
UR	Labour	Unemployment Rate
AI		Average Weekly Earnings
LC		Labour Cost Index
BDIR	Banking & Investments	Bank Deposit Interest Rate
BCIR		Bank Credit Interest Rate
I		Net Investment by UK financial institutions
FDI		Foreign Direct Investments
RS	Business Cycle	Retail Sales Index
IP		Industrial Production Rate
HP		House Price Index
PMI		Purchasing Managers Index (composite)

- *Prices Indexes.* It now includes the FTSE100 index as a subclass. But the Ontology can be supplemented with other market indexes if the user needs it.

- *Macroeconomic Indicators.* At the moment, this class contains 16 indictors that affect market indexes.

- *National Statistics Indicators.* This is the upper class of the indicators as compared to class Macroeconomic Indicators. It consists of indicators provided from which macroeconomic indicators are calculated – for example, Population figures which is the basis for Unemployment Rate calculation.

- *Changes in Government Regulations* is a special class that affects both Macroeconomic Indicators and National Statistics Indicators classes. It includes the UK Fiscal Policy, Monetary Policy, Labour Policy changes, etc.

The OMIP Ontology is a prototype version, and it can be updated in future with the other factors affecting the changes in market index price. The Full Ontology hierarchy developed in the Protege environment is shown in Fig.16. The more detailed OMIP Ontology diagram is given in Appendix D.

Besides, SubClassOf structure the Ontology includes three types of non-hierarchical connections: *hasEffectOn* (e.g. sets the relationships between Macroeconomic Indicators and Market Indexes), *directlyRelatedTo* and *indirectlyRelatedTo* (e.g. establishes the relationships between Macroeconomic Indicators and National Statistics Indicators).

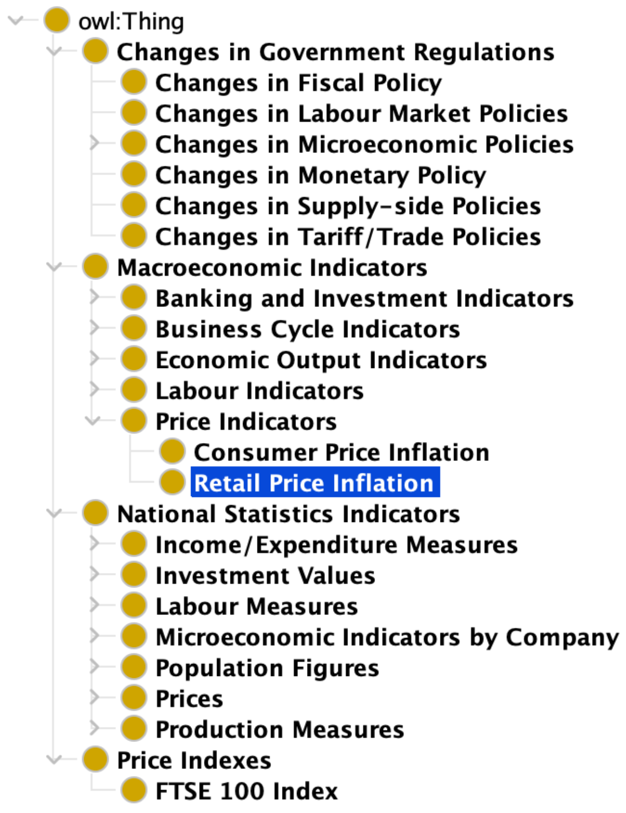

owl:Thing
Changes in Government Regulations
Changes in Fiscal Policy
Changes in Labour Market Policies
Changes in Microeconomic Policies
Changes in Monetary Policy
Changes in Supply-side Policies
Changes in Tariff/Trade Policies
Macroeconomic Indicators
Banking and Investment Indicators
Business Cycle Indicators
Economic Output Indicators
Labour Indicators
Price Indicators
Consumer Price Inflation
Retail Price Inflation
National Statistics Indicators
Income/Expenditure Measures
Investment Values
Labour Measures
Microeconomic Indicators by Company
Population Figures
Prices
Production Measures
Price Indexes
FTSE 100 Index

Figure 16: Protégé Environment: Full Ontology of Market Index Prediction

In Fig.17 one can see the 'Usage' tab of the Retail Price Inflation (RPI) indicator in Protege. It shows that the entity contains three data attributes: *Weight* (which is the indicator importance coefficient), *Value* (the actual value of the Ratio) and *Date* (time-slot). The rest of the indicators and the market index contain the same data attributes. Also, this tab shows the relations of the indictor with other entities in the Ontology. Moreover, every entity includes information about synonyms and term definitions.

An LSTM file containing the full version of OMIP Ontology – *Price_Index_Prediction_v3.2.owl* (see this file attached to the paper) is used to create the skeleton of the MIPCM model Graph DB in Neo4j.

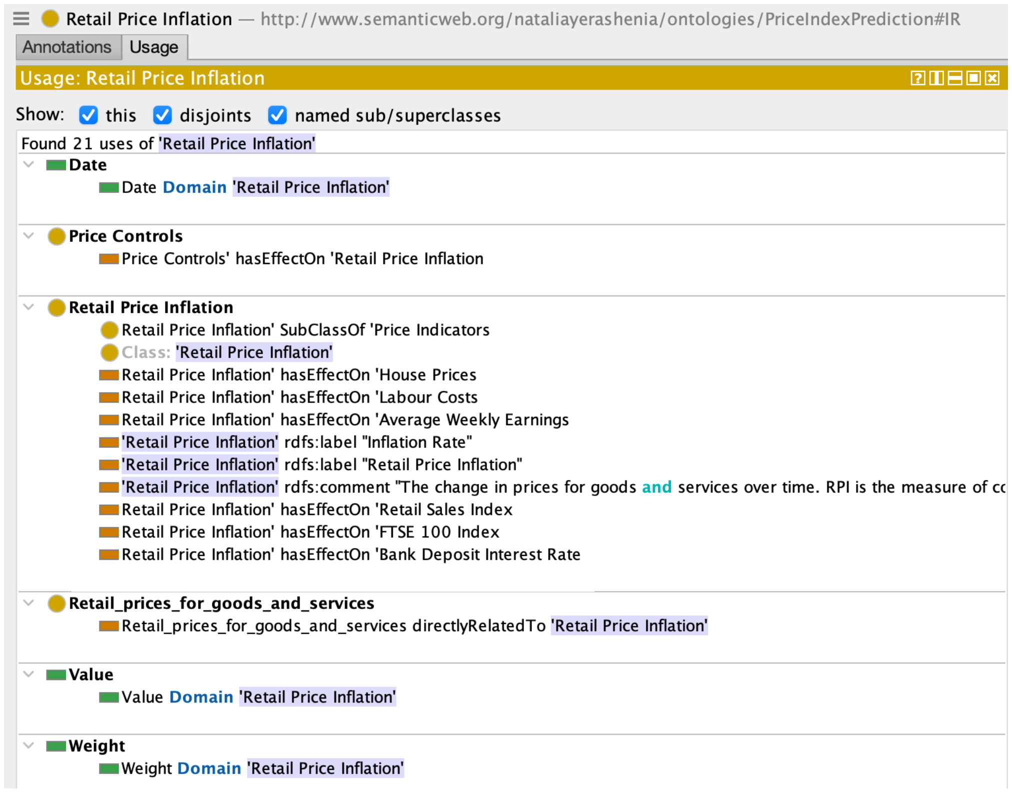
5.3 Developing Semantic Graph Database

The first step in creating a Neo4j Graph DB for Market Index Prediction is to transfer the OMIP Ontology LSTM file made in Protégé into Neo4j.

A separate file can be created in the Python environment, where the code for transportation will be placed – see the file *import_to_neo4j_uc2.py* attached.

First, I should create and open an empty graph called *'FTSE100'* (the name doesn't matter, though) in the Neo4j app window. Then define the LAN connection parameters to connect to the Graph DB in Python.

Now I need to import the LSTM file itself (the owlready2 library will help us with this). After that, I can proceed directly to extracting information from the LSTM ontology and using it as a base for building



Retail Price Inflation — http://www.semanticweb.org/nataliayerashenia/ontologies/PriceIndexPrediction#IR

Annotations | Usage

Usage: Retail Price Inflation

Show: ☑ this ☑ disjoints ☑ named sub/superclasses

Found 21 uses of 'Retail Price Inflation'

- **Date**
  - Date Domain 'Retail Price Inflation'
- **Price Controls**
  - Price Controls' hasEffectOn 'Retail Price Inflation
- **Retail Price Inflation**
  - Retail Price Inflation' SubClassOf 'Price Indicators
  - Class: 'Retail Price Inflation'
  - Retail Price Inflation' hasEffectOn 'House Prices
  - Retail Price Inflation' hasEffectOn 'Labour Costs
  - Retail Price Inflation' hasEffectOn 'Average Weekly Earnings
  - 'Retail Price Inflation' rdfs:label "Inflation Rate"
  - 'Retail Price Inflation' rdfs:label "Retail Price Inflation"
  - 'Retail Price Inflation' rdfs:comment "The change in prices for goods and services over time. RPI is the measure of cc
  - Retail Price Inflation' hasEffectOn 'Retail Sales Index
  - Retail Price Inflation' hasEffectOn 'FTSE 100 Index
  - Retail Price Inflation' hasEffectOn 'Bank Deposit Interest Rate
- **Retail_prices_for_goods_and_services**
  - Retail_prices_for_goods_and_services directlyRelatedTo 'Retail Price Inflation'
- **Value**
  - Value Domain 'Retail Price Inflation'
- **Weight**
  - Weight Domain 'Retail Price Inflation'

Figure 17: Protégé Environment: Retail Price Inflation Usages in PIPO Ontology

a graph.

```
greeter = OntotologyToGraphDatabase("bolt://localhost:7687", "neo4j", "11111")
onto_path.append(".")
onto = get_ontology("Price_Index_Prediction_v3.2.owl")
onto.load()
```

Nodes and Relationships are created in Python using the Cypher queries.

All elements of the ontology are transferred as graph nodes. In this case, it is necessary to get rid of all prefixes in the names of entities, which are automatically added to the LSTM file. Then hierarchical relationships between entities are added, and then the relationships that are hidden in the annotations of the ontology (non-hierarchical). Thus, an *Ontology template graph* is formed in the environment (see Definition 3.3).

To convert an Ontology template graph to an *Ontology full graph* (see Definition 3.2), a graph that is already filled with data, I need to use data from a CSV file with data taken from an external source – *Processed_Input_Data_FTSE100_1985_21.csv*. So the first step is to export the file itself to the Python environment, then correlate the names of the created nodes with the names of the columns in the file.

The complexity of this Use Case is that the input data is presented as time-series (I have 430 monthly time-slots). This means that Graph DB should contain 430 variations of the Ontology full graph (the will be differentiated by ”Date” data property).

My software solution reflects the limitations of the Neo4j environment. Neo4j environment doesn’t provide any built-in tools to present the time-series data in a more convinient way. Official Neo4j website offers to use KeyLines plug-in[32], but unfortunately it is not avaliable for academic purposes.

Each FTSE100 and Macroeconomic Indictors nodes in Neo4j should contain: ”Value”, taken from the external source, and ”Date” (for example, 01/10/2020 – the dataset’s last time-slot). The CSV file dataset (*Processed_Input_Data_FTSE100_1985_21.csv*) is formed using the information from *Office of National Statistics* official website[33] and market indexes database on *Yahoo.Finance* website[34].

```
    processed_input_csv_dates =
pd.read_csv("Processed_Input_Data_FTSE100_1985_21.csv").values[:, 0]
greeter.reset_graphs()
    add_nodes_query = ""
    match_nodes_queries = ""
    create_edges_queries = ""
    i = 0
    row = 0
    for date in processed_input_csv_dates:
        add_nodes_query = greeter.add_nodes_in_neo4j(onto, date, row,
        add_nodes_query)
        row += 1
        i += 1
        if (i % 10 == 0):
            with greeter.driver.session() as session:
                session.run(add_nodes_query)
            add_nodes_query = ""
    i = 0
    row = 0
    for date in processed_input_csv_dates:
        greeter.add_annotation_edges_in_neo4j(onto, date, row)
        greeter.add_subclass_edges_in_neo4j(onto, date, row)
        row += 1
        i += 1
    greeter.close()
```

To save the necessary data from the Graph DB for transferring to other model components the separate file with Python code is used – *input_data_import_from_neo4j.py*; the data is saved in a separate new CSV

[32]https://neo4j.com/blog/graphs-in-time-and-space/
[33]https://www.ons.gov.uk/
[34]https://finance.yahoo.com/quote/%5EFTSE?p=%5EFTSE

file called *Ratios_Export.csv.*

After the first iteration of the model, the system generates a CSV file with the weights of the model input indicators called *weights_importance_uc2.csv* (see below how this file is created). After exporting and processing this file I form the next "layer" of the Graph DB - *Ontology full graph with weights* (see Definition 3.1).

```
weights = pd.read_csv("weights_importance_uc2.csv", header=0).values
weights = weights.T
weights_columns = np.array(weights[0]).tolist()
weights_values = np.array(weights[1]).tolist()


    for row in processed_input:
    match_nodes_query = "MATCH "
    set_nodes_values_query = "SET "
    set_nodes_weights_query = "SET "
    complete_query = ""
    date = row[0]
    for c in onto.classes():
        if str(c.name) in processed_input_columns:
            match_nodes_query += "({cname}:Class {{name: '{cname}',
            Date: '{date}'}}), "\.format(cname=c.name,
                        date=date)
            index_of_class = processed_input_columns.index(c.name)
            set_nodes_values_query += "{cname}.Value = {csv_value}, "\
                .format(cname=c.name,
                       csv_value=row[index_of_class])
        if (str(c.name) in weights_columns) & (date == processed_input[-1][0]):
            index_of_weight = weights_columns.index(c.name)
            set_nodes_weights_query += "{cname}.Weight = {csv_weight}, " \
                .format(cname=c.name,
                        csv_weight=weights_values[index_of_weight])
    match_nodes_query = match_nodes_query[:-2]
    set_nodes_values_query = set_nodes_values_query[:-2]
    if (date == processed_input[-1][0]):
        set_nodes_weights_query = set_nodes_weights_query[:-2]
    if (date == processed_input[-1][0]):
        complete_query = match_nodes_query + " " +
        set_nodes_values_query + " " + set_nodes_weights_query + " RETURN *"
    else:
        complete_query = match_nodes_query + " " +
        set_nodes_values_query + " RETURN *"
    print(match_nodes_query)
    print(set_nodes_values_query)
    print(set_nodes_weights_query)
    print(complete_query)
    with greeter.driver.session() as session:
        session.run(complete_query)
greeter.close()
```

The full Ontology of Market Index Prediction graph with weights, which built in Neo4j environment is given in Appendix E.

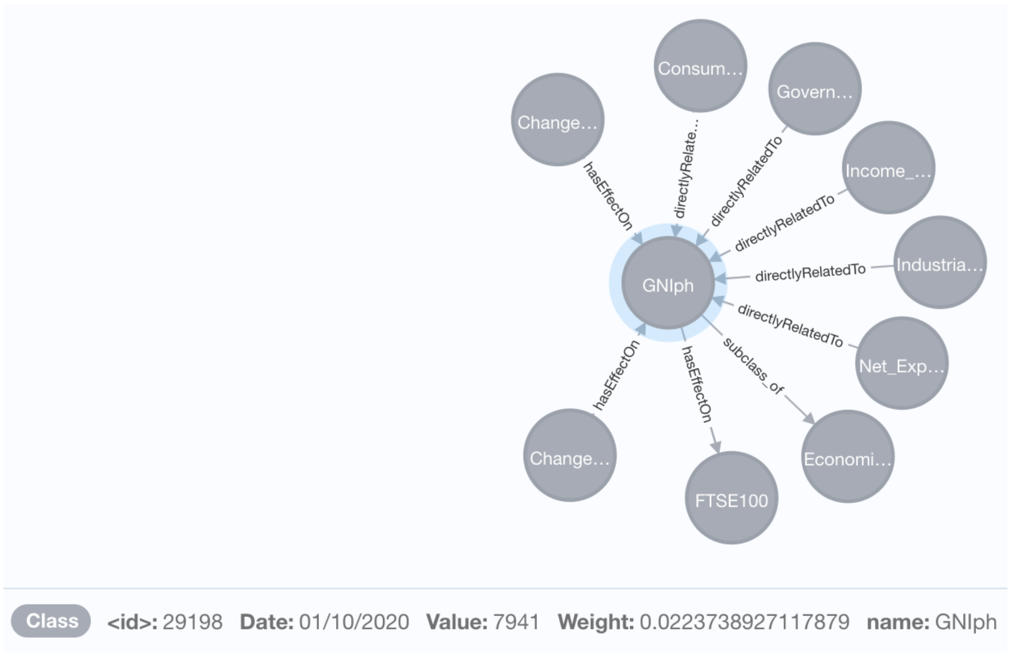
Next, based on the weights obtained, I can proceed with the *Feature Selection*.

Feature Selection using Graph Database. The following separate file called *cypher_queries_uc2.py* in the Python environment is responsible for it. Then, through certain queries from several indicators in one category, one with the maximum weight is selected. At the same time, should Neo4j display a graph containing only the indicators remaining after the selection. The data of this graph is exported as a CSV file for transporting it to the next component of the system.

According to Table 5 the Macroeconomic Indicators in MIPCM Model are divided into 5 categories. For example, the feature selection Cypher query for Economic Output Indicators category should be present as:

```
 economic_output_indicators_level_2_mw_node = session.run("""MATCH (n1:Class
 {name: "Economic_Output_Indicators_Level_2", Date: "01/10/2020"})<-
 [r1:subclass_of]-(scn1:Class)
               WITH max(scn1.Weight) AS maximum
               MATCH (n2:Class {name: "Economic_Output_Indicators_Level_2",
               Date: "01/10/2020"})<-[r2:subclass_of]-(scn2:Class)
               WHERE scn2.Weight = maximum
               RETURN scn2""").data()
...
eoi_l2_mw_node_name = economic_output_indicators_level_2_mw_node[0]["scn2"]
["name"]
...
max_weighted_features = [
    ...,
    ...,
    ...,
    eoi_l2_mw_node_name,
...]
```

The result of this query is shown in Fig.18.



Class <id>: 29198 Date: 01/10/2020 Value: 7941 Weight: 0.0223738927117879 name: GNIph

Figure 18: Neo4j Environment: Returning the Macroeconomic Indicator with the maximum weight in Economic Output Indicators category

Then the structured and selected features (indicators) are transferred to Python environment as CSV file called *feature_selection_from_neo4j.csv*, which is served as input data for NN.

5.4 Developing Machine Learning Engine: Hybrid Engine

As part of a computational model for FTSE100 prediction, I developed a complex Machine Learning Engine (ML Engine) – *Hybrid Engine* .

In the context of my thesis, *a Hybrid Machine Learning Engine (Hybrid Engine)* is a multi-component predictive tool that combines several types of machine learning models to leverage their unique strengths for improved prediction outcomes. It consists of parallel processing Long Short-Term Memory Network units, known for their proficiency with time-series data, alongside another predictive model – Linear Regression. These models work in synergy, with the output of the parallel processing units combined, through a concatenation unit to serve as an input for the final predictive unit, thereby enhancing the engine's overall predictive accuracy, efficiency, and adaptability to complex tasks.

By integrating the outputs of several LSTM networks, the hybrid MLE can potentially capture more nuances and patterns in the data. This can lead to more accurate and reliable predictions.

In this use case, Hybrid Engine consists of 17 parallel LSTMs for the market index and 16 Macroeconomic Indicators, a Concatenation unit and the LR unit, which analyse the LSTMs merged results.

I use this method as the basic tool in this use case as it is considered to be one of the most powerful for market data analysis [118], [120], [115], etc.

The main feature of the ML Engine is a *Concatenation Unit* which merges the parallel LSTMs' output into a single time-series dataset, which is used as input for Linear Regression (LR) Unit. Meanwhile, Linear Regression Unit produces the final result.

The Architecture of this Hybrid ML Engine for MIPCM is presented in Fig. 19 (the expanded version is in Appendix F).

As the basis for the Python code, I use *TensorFlow* and *Keras* libraries, which allow us to create machine learning models quickly and easily.

Collecting the Data. For LSTMs training, it is necessary to prepare a set of training data: examples of input data and their corresponding outputs.

As the training data, I use a monthly closing price and indexes values starting from January 1985. So, each LSTM and Macroeconomic Indicator input dataset has a vector form. The output of each LTSM is the particular feature's values for the recent 60 months (the last date of the dataset is October 2020), also presented as a vector. Now I can compare the result with the real data to check the accuracy of LSTMs' predictions.

I used *Office of National Statistics* official website and market indexes database on *Yahoo.Finance* to collect the data needed and save it as *Processed_Input_Data_FTSE100_1985_21.csv* file; it contains 430 time-slots from January 1985 to October 2020 (see Appendix H).

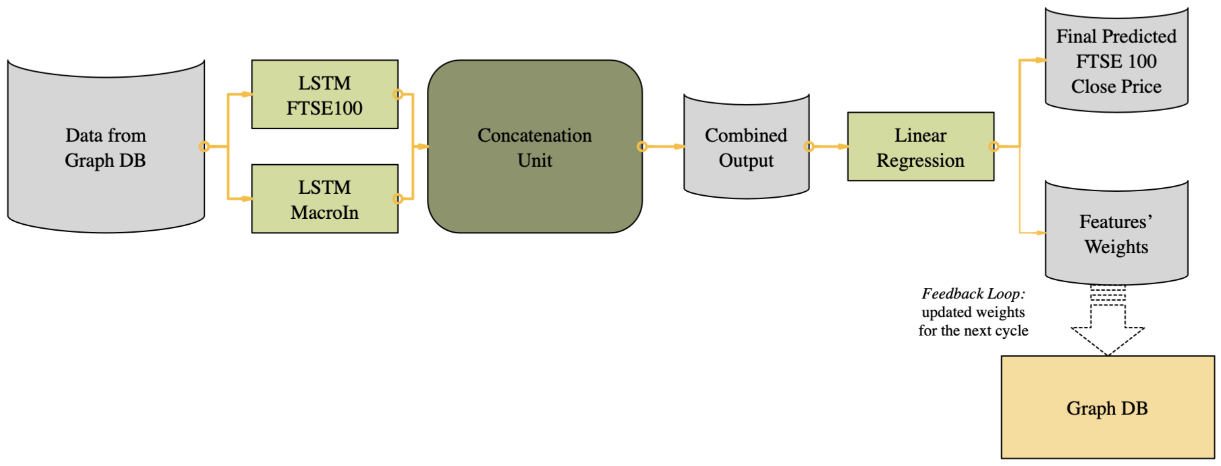


Figure 19: FTSE100 Machine Learning Engine Components and Dataflow

Pre-processing of Data. This part has already been done by means of the Semantic DS. See Sections 5.2 and 5.3.

Design, Training and Quality Assessment of a Hybrid Machine Learning Engine

Developing LSTMs. The first stage of creating a comprehensive ML Engine is developing a separate LSTM for each of the 17 features, plus one for the market index – FTSE100.

As mentioned before, one of the features of LSTM is that this approach can't be used for processing multiple time-series input. So I have to built an LSTM for each of the features separately and then concatenate the results.

All 17 LSTMs has a similar code, so the FTSE100 LSTM is presented below as an example.

Loading and normalising the input dataset:

```
dataset_train = pd.read_csv('Processed_Input_Data_FTSE100_1985_21.csv', header=0, index_col=0)
training_set = dataset_train.iloc[:, 0:1].values

sc = MinMaxScaler(feature_range=(0, 1))
training_set_scaled = sc.fit_transform(training_set)
```

Dividing the dataset into training and testing data, testing size is 60 months:

```
X_train = []
y_train = []
for i in range(0, (len(training_set) - testing_set_size)):
    X_train.append(training_set_scaled[i:i+1, 0])
    y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)

X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

Building a sequential LSTM Model:

```
model = Sequential()
model.add(LSTM(units = 64,input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.3))
model.add(Dense(1))

model.compile(loss='mse',optimizer='adam', metrics=['mean_squared_error'])
model.summary()

lstm_pred = model.fit(X_train,y_train,batch_size=30,epochs=40)

dataset_test = pd.read_csv('Processed_Input_Data_FTSE100_1985_21.csv', header=0, index_col=0)
real_stock_price = dataset_test.iloc[(len(training_set) - testing_set_size):, 0:1].values
```

```
dataset_total = pd.concat((dataset_train['FTSE100'], dataset_test['FTSE100']), axis=0)
inputs = dataset_total[len(dataset_total) - len(dataset_test):].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range((len(training_set) - testing_set_size), len(training_set)):
X_test.append(inputs[i:i+1, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

Using the model for data prediction:

```
predicted_stock_price = model.predict(X_test)
predicted_stock_price = np.reshape(predicted_stock_price, (testing_set_size, 1))
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

As it can be seen, I use *MinMaxScaler*, and *NumPy* .reshape functions to normalise and then denormalise the input data.

This simple sequential LSTM model includes 64 units; with the dropout rate set to 30%. The input shape equals the number of time-series slots for testing – 370 (which is 430-60). The density of the model is 1.

Then, the model is compiled. I used the 'Mean Squared Error' function as a loss function and 'Adam' (A Method for Stochastic Optimisation) as a replacement optimisation algorithm.

The parameters 'batch_size' and 'epochs' should be tuned for the particular dataset. To identify the optimal values for these parameters I use the *GRID Search CV*.

Finally, the model is trained using the .fit function, e.g., for FTSE100 – with 40 epochs and batch size 30.

The LSTMs are saved as separate files: *lstm_FTSE_100.py*, *lstm_GDP.py*, *lstm_GNIph.py*, *lstm_EG.py*, *lstm_IR.py*, *lstm_CPI.py*, *lstm_UR.py*, *lstm_AWE.py*,*lstm_LC.py*, *lstm_BDIR.py*, *lstm_BCIR.py*,*lstm_I.py*, *lstm_FDI.py*, *lstm_IP.py*, *lstm_RS.py*, *lstm_HP.py*, *lstm_PMI.py*.

The result of FTSE100 LSTM prediction for 60 last months (5 years) is visualised in Fig. 20. The MAPE of a Single LSTM model for FTSE100 close price prediction is recorded as 11.450% (see Table 6).

The output graphs of the rest LSTMs can be found in Appendix G.

Developing Concatenated Output. The next unit of the ML Engine is Concatenation. Using this NumPy function, I present the output vectors of 17 LSTMs as one joint matrix – combined output. The Python code runs from a separate file *main.py*.

Merging the parallel LSTM's outcomes into one:

```
lstm_predicted_stock_prices = [
    lstm_FTSE_100.predicted_stock_price,
```

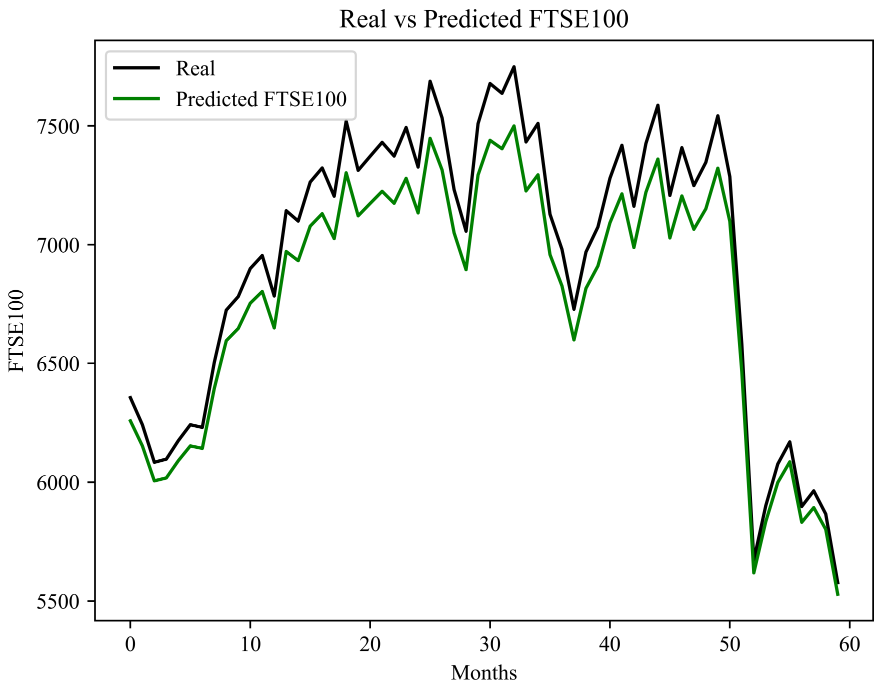


Figure 20: FTSE100 LSTM prediction

```
    lstm_GDP.predicted_stock_price,
    lstm_GNIph.predicted_stock_price,
    lstm_EG.predicted_stock_price,
    lstm_IR.predicted_stock_price,
    lstm_CPI.predicted_stock_price,
    lstm_UR.predicted_stock_price,
    lstm_AWE.predicted_stock_price,
    lstm_LC.predicted_stock_price,
    lstm_BDIR.predicted_stock_price,
    lstm_BCIR.predicted_stock_price,
    lstm_I.predicted_stock_price,
    lstm_FDI.predicted_stock_price,
    lstm_IP.predicted_stock_price,
    lstm_RS.predicted_stock_price,
    lstm_HP.predicted_stock_price,
    lstm_PMI.predicted_stock_price
]
number_of_lstms = len(lstm_predicted_stock_prices)
```

```
combined_output = np.concatenate(lstm_predicted_stock_prices, axis=1)
dataset_test = pd.read_csv('Processed_Input_Data_FTSE100_1985_21.csv', header=0,
index_col=0)
real_stock_price = dataset_test.iloc[0:, 0:1].values
real_stock_price = real_stock_price[len(real_stock_price) - testing_set_size:, 0:1]
combined_output = np.concatenate([combined_output, real_stock_price], axis=1)
```

The combined output can be saved as a CSV file called *combined_output.csv*, which will be used as input data for the LR unit of ML Engine. This CSV file contains

- 18 columns – LSTM Predicted FTSE100 index, 16 LSTM Predicted MacroIn, Real FTSE100 index
- 60 rows (time-slots).

However, the features still need to rely on each other at this stage.

Linear Regression. To calculate the joint result, the final most accurate prediction of FTSE100, considering the other 16 features, I use Linear Regression (LR) and implement it in Python using the SKLearn library.

First, the combined output should be split into two parts. The first (the oldest) time-slots of 17 input features plus the actual values of FTSE100 for the same time-slots are used as the training data for the LR. Accordingly, the LR analyses the dependency of the features from the actual FTSE100 price. The last (the newest) time-slots, e.g. last 12 months, are the LR input data (for testing).

Defining the dataset:

```
combined_output = pd.read_csv('combined_output.csv', header=0)
print(combined_output)
X, y = combined_output.iloc[0:testing_set_size-12,
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]].values,
combined_output.iloc[0:testing_set_size-12, -1].values
print(X)
```

Defining and fitting the Linear Regression model:

```
model = LinearRegression()
model.fit(X, y)
```

Using the model for data prediction:

```
test_X = combined_output.iloc[testing_set_size-12:,
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]].values
prediction = model.predict(test_X)
print(f'Prediction: {prediction}')
```

The model is tested using Mean Absolute Percentage Error (MAPE) formula to compare the accuracy of the FTSE100 index predicted by LSTM only and by LR using multiple features. Once LR passes the testing, the ML Engine is ready to use.

Before performing Feature Selection, the MAPE of the MIPCM model is 0.550%, which can be compared to the MAPE of a single LSTM at 11.450% (see Table 6).

Calculating the Weights for the Feedback Loop. After LR code for MIPCM Model (*linear_regression.py*) is executed the *feature importance analysis* of the input data should be made. For these purposes the *Random Forest Feature* method is used (the similar code used for Use Case 2, Section 4.4).

Finally, the result is saved as a CSV file (*weights_importance_uc1.csv*), which is further transferred to the Graph DB (as a part of the feedback loop).

5.5 Applying the Developed Market Index Prediction Computational Model for FTSE100 index

Once the dataset needed for the analysis is collected, it is exported to the template Neo4j Graph DB as the "Values" of the particular nodes.

Further, the file from Neo4j containing 430 time-slots of FTSE100 values, plus 16 MacroIn values (it can be called *Input_Data_from_Neo4j.csv* or *Processed_Input_Data_FTSE100_1985_21.csv*) should be imported to the Python environment (also see Appendix H).

After exporting these tree datasets to the ML Engine the following files should be run one-by-one:

- *main_lstms_concatenation_unit.py* – runs 17 LSTMs and generates a combined output
- *linear_regression.py* – runs LR using an LSTMs' combined output as input.

Then, the model should be tested using the MAPE formula to compare the accuracy of FTSE100 index predicted by LSTM only and by LR using multiple features, i.e. using Hybrid Engine. The result of this testing is presented in Fig. 21 and Table 6.

The plot shows the

It can be seen that the FTSE100 close price using complex Hybrid ML Engine prediction, using 17 LSTMs LR (green), is more accurate than the prediction results after LSTM (orange).

The comparison of prediction results indicates that the accuracy of FTSE100 close price prediction is higher when using the Hybrid ML Engine approach with 17 LSTM models and LR (represented in green) compared to the prediction results obtained from single LSTM model (represented in orange).

After the LR result is received, *feature importance analysis* can be preformed by using *feature_importance_uc2.py* to evaluate the weights of the features. Its output file *weights_importance_uc1.csv* can now be added to Neo4j Graph using a Feedback loop.

The Random Forest Feature Importance Analysis results for MIPCM are presented in Fig.22.

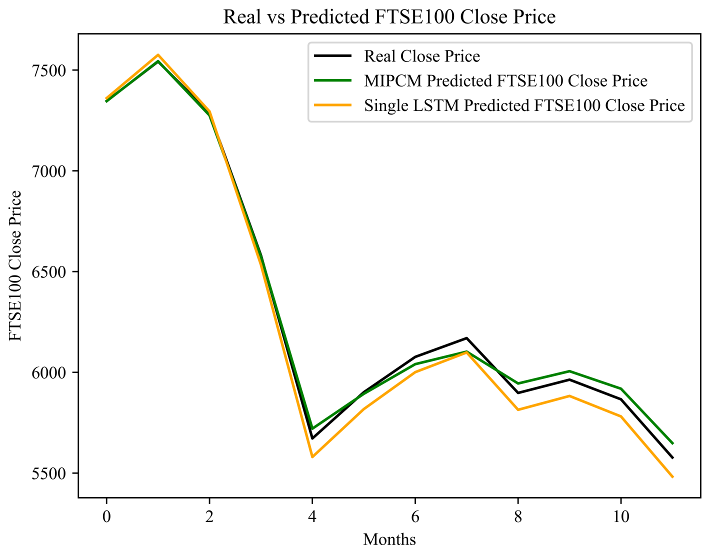


Figure 21: Linear Regression without Feature Selection Results: the Plot

Then, by means of Neo4j queries (*cypher_queries_uc2.py*), the weights are used to identify the 5 most valuable MacroIn (one for each category of indicators). As a result, the model picked the following ratios: CPI, I, IP, GNIph and UR.

The usage of only six inputs, FTSE100 close price plus five MacroIn, will prevent the Hybrid ML Engine from overfitting in future, providing more accurate outcomes. To prove this, I run the LR code (2^{nd} models' cycle) using as an input the five selected MacroIn only. The result of this testing is presented in Fig. 23 and Table 6.

After comparing the Final MAPE of MIPCM model without feature selection – 0.550, with the Final MAPE after the model's first cycle (includes feature selection) – 0.503 (see Table 6), I can state that the feature selection increased the efficiency of the MIPCM Model.

Comparing results of MIPCM model and ARIMAX model. To justify the efficacy of the formulated MIPCM model, a comparative assessment against the ARIMAX model is conducted.

The ARIMAX model is instantiated through the ARIMA method from the *statsmodels* Python library. The code is available on my GitHub[35].

The ARIMAX model is characterised by three primary parameters: the number of autoregressive terms, the number of differences taken, and the number of moving average terms. These parameters form the order argument in the ARIMA model instantiation. My model's parameters are set to (2, 0, 1), indicating that the model is an ARMA(2,1) model with two autoregressive terms and one moving average term. No differencing is applied, suggesting that the time series is stationary. The model also takes in an exogenous variable, *exog*, which can be any variable outside the dataset that might impact the outcome. X_train (the transformed

[35]`https://github.com/Yerashenia/Predictive-Computational-Model-PCM`

training data) is an exogenous variable in this case.

The performance of ARIMAX and MIPCM predictive models is evaluated using the MAPE.

Comparing the Single LSTM and ARIMAX models, which have MAPE values of 11.450% and 10.671%, respectively, it can be seen that the ARIMAX model slightly outperforms the LSTM model. This could be due to the ARIMAX model's ability to incorporate autoregressive and moving average components along with an exogenous variable, which can capture more complex dynamics in the data compared to LSTM.

In contrast, the MIPCM model (including Hybrid Engine) significantly outperforms LSTM and ARIMAX, regardless of whether feature selection is used. Without feature selection, the MIPCM model achieves an impressively low MAPE of 0.550%, and this value decreases to 0.503% when feature selection is applied.

These results highlight the superior performance of the MIPCM model in predicting the FTSE100 index, even without feature selection. The introduction of Neo4j feature selection further refines the model's performance, illustrating the value of employing appropriate feature selection techniques. This comparison underscores the premise that the right combination of machine learning methodologies and feature selection strategies can significantly enhance financial time series forecasting accuracy.

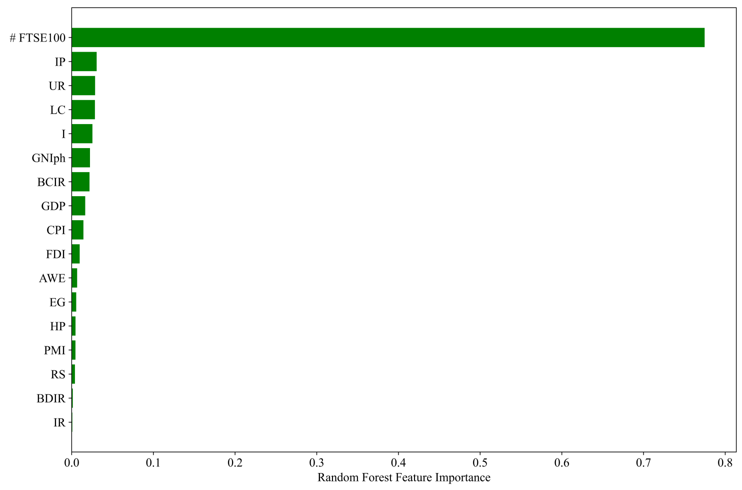


Figure 22: Random Forest Feature Importance Analysis Result for PIPCM features

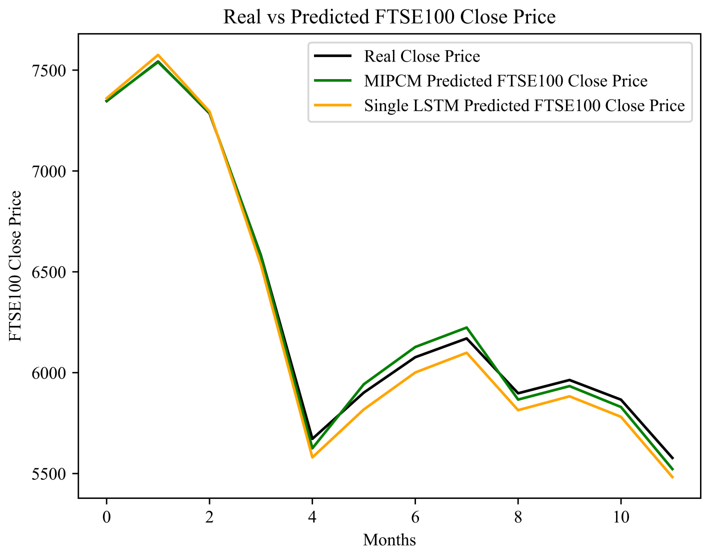


Figure 23: Linear Regression with Feature Selection Results: the Plot

Table 6: Market Index Prediction Computational Model output for FTSE100

Date	FTSE100 real close price	FTSE100 single LSTM prediction	FTSE100 prediction using MIPCM (without Feature Selection)	FTSE100 prediction using MIPCM (with Feature Selection)
01/11/2019	7346.5	7360.787	7347.7	7348.497
01/12/2019	7542.4	7575.345	7539.0	7540.152
01/01/2020	7286	7294.767	7302.8	7288.471
01/02/2020	6580.6	6533.874	6586.8	6576.441
01/03/2020	5672	5579.965	5593.2	5625.110
01/04/2020	5901.2	5817.636	5910.6	5942.516
01/05/2020	6076.6	6000.896	6132.8	6127.196
01/06/2020	6169.7	6098.643	6279.0	6223.647
01/07/2020	5897.8	5814.096	5835.6	5866.593
01/08/2020	5963.6	5882.697	5909.2	5933.417
01/09/2020	5866.1	5781.105	5794.5	5829.631
01/10/2020	5577.3	5482.374	5478.1	5522.119
	Mean Absolute Percentage Error, %	11.45	0.550	0.503

6 Conclusions and Future Work

Conclusions. I have developed a novel generic component-based architecture for *Predictive Computational Model* (see Section 3.4, Fig. 5), which integrates the Semantic Database System and a Machine Learning Engine.

Under the *Generic Architecture*, I understand an abstract framework with a generic structure with no physical realisation specification, i.e. its interpretation is not confined so far; it is a kind of template that defines the structure of the model and its data flow. It does not designate a specific domain usage, a type of data or a machine learning method which processes data.

The software system is divided into abstract elements of the structure, known as components, in the *Component-Oriented Approach.* These components are organised in a specific manner and are responsible for solving subtasks within the system's overall task. They interact with the entire system based on certain principles.

In my work, by *Computational Model*, I mean a set of interconnected and interacting computational procedures and methods designed to collect, store, process and distribute information.

I proposed the *KDD standard* as the basis for the development of the GA for PCM model applying the new semantic methods of information research and modification. Following these steps, I divided the process of performing my complex fundamental problem of analysing and forecasting data into the following stand-alone sub-tasks that are fulfiled by the components of my model: preparation of raw input data, structuring and storing data, feature selecting data for further analysis & formation of the new more efficient dataset, data analysis itself, data evaluation and results' improvement.

The Semantic Database System (Semantic DS) is a novel development, which comprises the *Ontology*, the *Graph Database* and the *Feature Selection* components, aiming to store heterogeneous data and structure and prepare it for further analysis.

Raw Data Interpreter collects raw data from the sources' standard database. The data gained are subsequently stored conveniently to support its efficient management. The interpreter allows feeding the data in an appropriate commonly used format (e.g., CSV) to the Graph Database module.

The *Ontology* defines a standard concept library for users and developers who need to share information in a domain. It includes machine-interpreted formulations of the basic concepts of the subject area and the relationship between them, making them more accessible for complex reasoning.

The *Graph Database* is the system's local database which stores the raw data exported from the external sources database, structuring it in the form of an interconnected graph based on the information taken from the Ontology. Employing Graph DB, the data can be adapted to the further system's components – Feature Selection and Machine Learning Engine. Additionally, Graph DB is dependent/dynamically updated by the feedback from the Machine Learning Engine (the core component of the system which returns the analytics outcomes). It allows Graph DB to keep the outcomes and analytical metadata (e.g. features' weights) from the models' previous iterations.

Feature Selection is optimised by Graph DB queries. Using the weights (importance coefficient) of the features calculated in the previous cycle of the model, the most valuable features can be selected using complex queries. The Graph DB allows generating the new input subset of features for the Machine Learning Engine (the next model's cycle).

Graph databases enable the formulation of the queries of any complexity handled by a dedicated query language (e.g., Cypher as part of Neo4j in my case). The efficiency of queries, though, is supposed to be improved by an expert.

Data analysis and forecasting using selected key features can be carried out through *Machine Learning Engine*, which can include a set of Data Analysis methods, such as Neural Networks, Regressions, Decision Trees, Support Vector Machines, etc. The ML Engine method choice depends on the particular input dataset characteristics and the specific nature of the analytical objectives (forecasting or classification).

Besides returning the outcome of the current iteration, ML Engine allows analysing of the weights of the input features and transfers this information to the Graph DB component by the feedback loop, enhancing the system performance during the subsequent iterations.

Thus, the research concentrated on applying a semantic approach for data pre-processing. As a result, an effective generic architecture solution integrating individual components of the 'Semantic Database System and Machine Learning Engine was developed and presented in three published papers: [1], [165], [176].

The developed system has been applied to analyse real-life data. The PCM model is implemented for two use cases: *Bankruptcy Prediction Computational Model* (see Section 4) and *Market Index Prediction Computational Model* (see Section 5) .

The data files and fully operating code for the use cases are developed, including unique Ontologies, Graph Databases, and Machine Learning Engines connected by the data transferring system and feedback loop. See Appendix H. The code is available on my GitHub[36].

The last stage of the research investigated the impact of multiple parameters related to predicting time-series data. The problem was complicated because LSTM, the primary and one of the most accurate methods of forecasting time-series data, is intended to analyse one input parameter only (dealing with time-series vectors). Therefore, to solve this unusual task, a unique hybrid Machine Learning Engine was developed, consisting of several parallel LSTMs, a Concatenation Unit, which transfers the LSTMs' output (in the form of a single time-series matrix) to a Linear Regression Unit, which produces the final result. The method is presented in Use Case 2 (see Section 5.4).

The experiments have shown the correctness and efficiency of the system. Furthermore, the underlying system's architecture is based on different components to foster flexibility and elasticity – existing components could be substituted by 'equivalent' elements that could also be suitable for other purposes.

The developed prototype is fully functioning and allows experimentation aimed at tuning and advancing the system. The proposed software solutions enable interested researchers (or even practitioners) to assemble similar systems following the presented methodology.

[36] `https://github.com/Yerashenia/Predictive-Computational-Model-PCM`

Revisiting the Research Objectives. With the *research's objectives* (refer to Section 1.2) serving as the guiding posts, this research has sought to expand the boundaries of computational models and data mining techniques. Hereafter, we reflect upon the objectives and examine the progress made concerning each.

The first research objective, examining and comparing the most popular existing data mining techniques, especially concerning financial/market data, was achieved through my extensive literature review and comparative analysis in Section 2.6. I explored different algorithms and techniques, focusing on their application in the financial and market domain. This process helped me understand the strengths, weaknesses, and practical considerations associated with each, providing a solid foundation for creating the Generic Architecture of PCM model.

The second objective, identifying the significant role of data pre-processing in data mining for interconnected and heterogeneous data, was fulfilled by delving into various semantic methods for data pre-processing. I explored the strengths of ontology (Section 2.4) and graph-based methods (Section 2.5) for handling complex, interrelated data structures, which allowed me to unify various data types and forms into homogeneous structure. This exploration was pivotal in forming the basis of my PCM, leading to a better understanding of managing vast and heterogeneous data effectively.

For the third objective, I investigated the components of a computational model for complex data mining problems and determined how information should flow within such a system. This investigation, provided in Section 3.3, informed the PCM model design, resulting in a model consisting of several integral components: Raw Data Interpreter, Ontology, Graph DB (including Feature Selection), and ML Engine. This design, stated in Section 3.4, also ensured a systematic data flow from initial collection to analysis, creating a smooth and effective process.

The fourth objective required the construction of a Semantic Database System (Semantic DS), which underpinned the PCM model. This was accomplished by initiating the Ontology component, where the use cases formulated two specific Ontologies. The Ontology component provided the foundational concepts for data analysis within the specified knowledge domain and organised the data through a semantic framework. This Ontology component was integrated with a Graph DB, facilitating data storage and pre-processing, thereby creating a unified Semantic DS. A significant element of this objective involved the conception of the Feature Selection component, tasked with pinpointing crucial indicators for comprehensively and accurately portraying the model's designated task. Detailed discussions about building the Semantic DS can be found in the chapters dedicated to the use cases.

The fifth objective involved the development of a Machine Learning Engine (ML Engine) capable of handling semantic data input. I explored various computational tools and evaluated their adequacy and accuracy for specific model purposes. The developed ML Engine was tested on real-life data, validating its capabilities in handling complex financial and market data.

The sixth objective was to develop a data exchange mechanism between the Semantic DS structural parts and other system components. I achieved this by creating a system architecture that allowed for automatic data transfer between components, ensuring a smooth flow of information throughout the system.

The seventh objective focused on creating a Feedback Loop for the model. This loop allowed me to transfer information about the importance of each input feature from the ML Engine to the Semantic DS, improving the efficiency of feature selection and overall output quality.

The final objective was to test the consistency and validate the entire model. I achieved this through rigorous testing of my PCM, using real-life data from two use cases: the UK companies' bankruptcy level detection and FTSE100 index prediction. The model demonstrated high reliability and accuracy, confirming its effectiveness in solving complex data mining problems. Thus, all of my outlined research objectives were satisfactorily achieved.

Future Work. This research opens the perspectives for the following studies:

- further research on using graph database for feature selection purposes, in particular, creating complex queries to generate more efficient data subsets for data mining;
- research on making a system fully automated;
- investigating the use cases involving Big Data analysis;
- research on the application of deep learning as the computational engine for the model;
- to supplement the Ontology of Bankruptcy Prediction and Ontology of Market Index Prediction with new components;
- research on adding the new components to the model, for example, the module that considers the environment's changes (external factors), such as a data crawling agent.

It should be noted that despite the apparent advantages of the presented model over other more simplified methods, it cannot be considered a self-contained application. However, it can be integrated as a plug-in into established analytical engines, for example, Apache Spark. Making the PCM model a self-contained application is also a part of future work.

Definitions

Acronyms

BPCM Bankruptcy Prediction Computational Model. 50, 53, 56, 57, 58, 61, 64, 68, 69, 70, 71, 72, 73, 74, 97

GA Generic Architecture. 38, 39, 41, 42, 45, 46, 51, 96

Graph DB Graph Database. 5, 7, 48, 50, 55, 61, 62, 64, 65, 72, 74, 75, 80, 81, 82, 83, 91, 96, 97, 98

KDD Knowledge Discovery in Databases. 9, 41, 42, 45, 96

LR Linear Regression. 7, 26, 27, 50, 85, 90, 91, 92

LSTM Long Short Term Memory. 7, 31, 32, 55, 61, 62, 80, 81, 85, 86, 87, 88, 90, 91, 93

MacroIn Macroeconomic Indicators. 90, 91, 92

MAPE Mean Absolute Percentage Error. 88, 91, 92, 93

MIPCM Market Index Prediction Computational Model. 77, 78, 80, 84, 86, 91, 92, 93, 97

ML Engine Machine Learning Engine. 6, 7, 8, 50, 55, 57, 71, 85, 86, 87, 88, 90, 91, 92, 97, 98, 99

NN Neural Network. 35, 65, 69, 70, 71, 72, 75, 85

OBP Ontology of Bankruptcy Prediction. 49, 57, 58, 59, 60, 61, 62, 65, 74, 78

OMIP Ontology of Market Index Prediction. 78, 79, 80

PCM Predictive Computational Model. 4, 5, 6, 8, 12, 22, 39, 42, 43, 45, 47, 50, 51, 52, 54, 55, 56, 77, 96, 97, 98, 99

RDF Resource Description Framework. 5, 19, 20, 21, 52, 53, 54, 55

RNN Recurrent Neural Networks. 31, 32

Semantic DS Semantic Database System. 5, 6, 7, 47, 69, 86, 96, 98, 99

References

[1] N. Yerashenia and A. Bolotov, “Computational modelling for bankruptcy prediction: Semantic data analysis integrating graph database and financial ontology,” in *2019 IEEE 21st Conference on Business Informatics (CBI)*, vol. 1. IEEE, 2019, pp. 84–93.

[2] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques.* Elsevier, 2011.

[3] U. Grimmer and H. Hinrichs, “A methodological approach to data quality management supported by data mining.” in *IQ*, 2001, pp. 217–232.

[4] Y. Roh, G. Heo, and S. E. Whang, “A survey on data collection for machine learning: a big data-ai integration perspective,” *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[5] M. Refaat, *Data preparation for data mining using SAS.* Elsevier, 2010.

[6] J. Huang, Y.-F. Li, and M. Xie, “An empirical analysis of data preprocessing for machine learning-based software cost estimation,” *Information and software Technology*, vol. 67, pp. 108–127, 2015.

[7] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, “A survey of machine learning for big data processing,” *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, p. 67, 2016.

[8] I. Taleb, R. Dssouli, and M. A. Serhani, “Big data pre-processing: A quality framework,” in *2015 IEEE international congress on big data.* IEEE, 2015, pp. 191–198.

[9] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, and F. Herrera, “Big data preprocessing: methods and prospects,” *Big Data Analytics*, vol. 1, no. 1, p. 9, 2016.

[10] C.-F. Tsai and J.-S. Chou, “Data pre-processing by genetic algorithms for bankruptcy prediction,” in *2011 IEEE International Conference on Industrial Engineering and Engineering Management.* IEEE, 2011, pp. 1780–1783.

[11] S. I. Koval, “Data preparation for neural network data analysis,” in *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus).* IEEE, pp. 898–901.

[12] G. G. Dumancas and G. A. Bello, “Comparison of machine-learning techniques for handling multicollinearity in big data analytics and high-performance data mining,” in *SC15: The International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 41–42.

[13] N. Mohd Nawi, W. H. Atomia, and M. Z. Rehman, “The effect of data pre-processing on optimized training of artificial neural networks,” 2013.

[14] L. Yu, S. Wang, and K. K. Lai, “Data preparation in neural network data analysis,” *Foreign-Exchange-Rate Forecasting With Artificial Neural Networks*, pp. 39–62, 2007.

[15] ——, “An integrated data preparation scheme for neural network data analysis,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 2, pp. 217–230, 2005.

[16] J. Cai, J. Luo, S. Wang, and S. Yang, "Feature selection in machine learning: A new perspective," *Neurocomputing*, vol. 300, pp. 70–79, 2018.

[17] A. Jain and D. Zongker, "Feature selection: Evaluation, application, and small sample performance," *IEEE transactions on pattern analysis and machine intelligence*, vol. 19, no. 2, pp. 153–158, 1997.

[18] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in *2014 science and information conference.* IEEE, 2014, pp. 372–378.

[19] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[20] A. Jović, K. Brkić, and N. Bogunović, "A review of feature selection methods with applications," in *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO).* Ieee, 2015, pp. 1200–1205.

[21] U. Stańczyk, "Feature evaluation by filter, wrapper, and embedded approaches," in *Feature Selection for Data and Pattern Recognition.* Springer, 2015, pp. 29–44.

[22] S. Lei, "A feature selection method based on information gain and genetic algorithm," in *2012 international conference on computer science and electronics engineering*, vol. 2. IEEE, 2012, pp. 355–358.

[23] I. S. Thaseen and C. A. Kumar, "Intrusion detection model using fusion of chi-square feature selection and multi class svm," *Journal of King Saud University-Computer and Information Sciences*, vol. 29, no. 4, pp. 462–472, 2017.

[24] M. A. Hall *et al.*, "Correlation-based feature selection for machine learning," 1999.

[25] N. El Aboudi and L. Benhlima, "Review on wrapper feature selection approaches," in *2016 International Conference on Engineering & MIS (ICEMIS).* IEEE, 2016, pp. 1–5.

[26] Y. Saeys, T. Abeel, and Y. Van de Peer, "Robust feature selection using ensemble feature selection techniques," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II 19.* Springer, 2008, pp. 313–325.

[27] V. Fonti and E. Belitser, "Feature selection using lasso," *VU Amsterdam research paper in business analytics*, vol. 30, pp. 1–25, 2017.

[28] M. B. Kursa and W. R. Rudnicki, "The all relevant feature selection using random forest," *arXiv preprint arXiv:1106.5112*, 2011.

[29] D. Dou, H. Wang, and H. Liu, "Semantic data mining: A survey of ontology-based approaches," in *Proceedings of the 2015 IEEE 9th international conference on semantic computing (IEEE ICSC 2015).* IEEE, 2015, pp. 244–251.

[30] C. Sirichanya and K. Kraisak, “Semantic data mining in the information age: A systematic review,” *International Journal of Intelligent Systems*, vol. 36, no. 8, pp. 3880–3916, 2021.

[31] G. P. Mansukhlal and C. Malathy, “Semantic integration with ontology based approach,” in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. IEEE, 2016, pp. 2257–2260.

[32] P. Castells, B. Foncillas, R. Lara, M. Rico, and J. L. Alonso, “Semantic web technologies for economic and financial information management,” in *European Semantic Web Symposium*. Springer, 2004, pp. 473–487.

[33] R. S. Tsay, *Analysis of financial time series*. John wiley & sons, 2005, vol. 543.

[34] D. Kim and J. C. Francis, *Modern portfolio theory: Foundations, analysis, and new developments*. John Wiley & Sons, 2013.

[35] L. Zhao and R. Ichise, “Ontology integration for linked data,” *Journal on Data Semantics*, vol. 3, pp. 237–254, 2014.

[36] Y. Hilpisch, *Python for finance: mastering data-driven finance*. O’Reilly Media, 2018.

[37] M. Gagnon, “Ontology-based integration of data sources,” in *2007 10th International Conference on Information Fusion*. IEEE, 2007, pp. 1–8.

[38] J. Pokornỳ, “Graph databases: their power and limitations,” in *IFIP International Conference on Computer Information Systems and Industrial Management*. Springer, 2015, pp. 58–69.

[39] L. Zhao, R. Ichise, Y. Sasaki, Z. Liu, and T. Yoshikawa, “Fast decision making using ontology-based knowledge base,” in *2016 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2016, pp. 173–178.

[40] R. Studer, V. R. Benjamins, and D. Fensel, “Knowledge engineering: Principles and methods,” *Data & knowledge engineering*, vol. 25, no. 1-2, pp. 161–197, 1998.

[41] J. De Bruijn, “Using ontologies-enabling knowledge sharing and reuse on the semantic web,” 2003.

[42] A. Arbaaeen and A. Shah, “Ontology-based approach to semantically enhanced question answering for closed domain: A review,” *Information*, vol. 12, no. 5, p. 200, 2021.

[43] B. Smith and W. Ceusters, “Ontological realism: A methodology for coordinated evolution of scientific ontologies,” *Applied ontology*, vol. 5, no. 3-4, pp. 139–188, 2010.

[44] T. Berners-Lee, J. Hendler, and O. Lassila, “The semantic web,” *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.

[45] N. Guarino, “Formal ontology, conceptual analysis and knowledge representation,” *International journal of human-computer studies*, vol. 43, no. 5-6, pp. 625–640, 1995.

[46] D. E. O'Leary, "Using ai in knowledge management: Knowledge bases and ontologies," *IEEE Intelligent Systems and Their Applications*, vol. 13, no. 3, pp. 34–39, 1998.

[47] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *International journal of human-computer studies*, vol. 43, no. 5-6, pp. 907–928, 1995.

[48] S. Moiseyenko and V. Ermolayev, "Conceptualizing and formalizing requirements for ontology engineering." in *PhD@ ICTERI*, 2018, pp. 35–44.

[49] R. Khosla and N. Ichalkaranje, *Design of intelligent multi-agent systems: human-centredness, architectures, learning and adaptation.* Springer, 2013.

[50] H. Tang and L. Song, "Ontologies in financial services: Design and applications," in *2011 International Conference on Business Management and Electronic Information*, vol. 5. IEEE, 2011, pp. 364–367.

[51] K. Schwarz, "Domain model enhanced search-a comparison of taxonomy, thesaurus and ontology," *unpublished thesis, University of Utrecht, available at: http://homepages. cwi. nl/˜ media/publications/masterthesis_kat_domainmodel_200*, vol. 5, 2005.

[52] C. Brewster and Y. Wilks, "Ontologies, taxonomies, thesauri: Learning from texts," 2004.

[53] J. Ye, G. Stevenson, and S. Dobson, "A top-level ontology for smart environments," *Pervasive and mobile computing*, vol. 7, no. 3, pp. 359–378, 2011.

[54] J. G. Gammack, "Different techniques and different aspects on declarative knowledge," in *Knowledge acquisition for expert systems.* Springer, 1987, pp. 137–163.

[55] S. Decker, "On domain-specific declarative knowledge representation and database languages," in *KRDB-98—Proceedings of the 5th Workshop Knowledge Representation meets DataBases, Seattle, WA*, 1998.

[56] I. Horrocks, "Owl: A description logic based ontology language," in *International conference on principles and practice of constraint programming.* Springer, 2005, pp. 5–8.

[57] T. R. Gruber, "The role of common ontology in achieving sharable, reusable knowledge bases," *KR*, vol. 91, pp. 601–602, 1991.

[58] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys (CSUR)*, vol. 40, no. 1, p. 1, 2008.

[59] V. Geroimenko and C. Chen, *Visualizing the Semantic Web: XML-based internet and information visualization.* Springer Science & Business Media, 2006.

[60] F. Antoniazzi and F. Viola, "Rdf graph visualization tools: A survey," in *2018 23rd Conference of Open Innovations Association (FRUCT).* IEEE, 2018, pp. 25–36.

[61] D. L. McGuinness, F. Van Harmelen *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, no. 10, p. 2004, 2004.

[62] B. Le Grand and M. Soto, "Topic maps, rdf graphs, and ontologies visualization," in *Visualizing the Semantic Web.* Springer, 2006, pp. 59–79.

[63] N. Noy, D. L. McGuinness *et al.*, "Ontology development 101," *Knowledge Systems Laboratory, Stanford University*, vol. 2001, 2001.

[64] R. J. Trudeau, *Introduction to graph theory.* Courier Corporation, 2013.

[65] A. Dickson, "Introduction to graph theory," *CRC Pres*, 2006.

[66] R. J. Wilson, *Introduction to graph theory, 4th edition.* Prentice Hall, 1998.

[67] V. N. Gudivada, D. Rao, and V. V. Raghavan, "Nosql systems for big data management," in *2014 IEEE World congress on services.* IEEE, 2014, pp. 190–197.

[68] Z. J. Zhang, "Graph databases for knowledge management," *IT Professional*, vol. 19, no. 6, pp. 26–32, 2017.

[69] S. Jouili and V. Vansteenberghe, "An empirical comparison of graph databases," in *2013 International Conference on Social Computing.* IEEE, 2013, pp. 708–715.

[70] V. Kolomičenko, M. Svoboda, and I. H. Mlỳnková, "Experimental comparison of graph databases," in *Proceedings of International Conference on Information Integration and Web-based Applications & Services.* ACM, 2013, p. 115.

[71] S. S. Devi and Y. Radhika, "A survey on machine learning and statistical techniques in bankruptcy prediction," *International Journal of Machine Learning and Computing*, vol. 8, no. 2, 2018.

[72] R. Bhutta and A. Regupathi, "Predicting corporate bankruptcy: Lessons from the past," *Asian Journal of Multidisciplinary Studies*, vol. 8, p. 1, 2020.

[73] W. H. Beaver, "Financial ratios as predictors of failure," *Journal of accounting research*, pp. 71–111, 1966.

[74] E. I. Altman, "Financial ratios, discriminant analysis and the prediction of corporate bankruptcy," *The journal of finance*, vol. 23, no. 4, pp. 589–609, 1968.

[75] R. J. Taffler, "The assessment of company solvency and performance using a statistical model," *Accounting and Business Research*, vol. 13, no. 52, pp. 295–308, 1983.

[76] J. A. Ohlson, "Financial ratios and the probabilistic prediction of bankruptcy," *Journal of accounting research*, pp. 109–131, 1980.

[77] W. Brinda and W. Brinda, "Least-squares linear regression," *Visualizing Linear Models*, pp. 39–61, 2021.

[78] R. Y. Nivetha and C. Dhaya, "Developing a prediction model for stock analysis," in *2017 International Conference on Technical Advancements in Computers and Communications (ICTACC)*. IEEE, 2017, pp. 1–3.

[79] D. Maulud and A. M. Abdulazeez, "A review on linear regression comprehensive in machine learning," *Journal of Applied Science and Technology Trends*, vol. 1, no. 4, pp. 140–147, 2020.

[80] D. Bhuriya, G. Kaushal, A. Sharma, and U. Singh, "Stock market predication using a linear regression," in *2017 international conference of electronics, communication and aerospace technology (ICECA)*, vol. 2. IEEE, 2017, pp. 510–513.

[81] R. Gomila, "Logistic or linear? estimating causal effects of experimental treatments on binary outcomes using regression analysis." *Journal of Experimental Psychology: General*, 2020.

[82] D. Alaminos, A. del Castillo, and M. Á. Fernández, "A global model for bankruptcy prediction," *PloS one*, vol. 11, no. 11, 2016.

[83] E. Ul Hassan, Z. Zainuddin, and S. Nordin, "A review of financial distress prediction models: logistic regression and multivariate discriminant analysis," *Indian-Pacific Journal of Accounting and Finance*, vol. 1, no. 3, pp. 13–23, 2017.

[84] K. Valaskova, T. Kliestik, L. Svabova, and P. Adamko, "Financial risk measurement and prediction modelling for sustainable development of business entities using regression analysis," *Sustainability*, vol. 10, no. 7, p. 2144, 2018.

[85] G. E. Box and G. JENKINS, "Gwilym m," *Time Series Analysis: Forecasting and Control. Revised Edition. Oakland, California: Editorial Holden-Day*, 1976.

[86] D. C. Montgomery, L. A. Johnson, and J. S. Gardiner, *Forecasting and time series analysis.* McGraw-Hill Companies, 1990.

[87] A. Raudys, V. Lenčiauskas, and E. Malčius, "Moving averages for financial data smoothing," in *Information and Software Technologies*, T. Skersys, R. Butleris, and R. Butkiene, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 34–45.

[88] J. H. Cochrane, "Time series for macroeconomics and finance," *Manuscript, University of Chicago*, vol. 15, p. 16, 2005.

[89] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[90] Z. Li, J. Han, and Y. Song, "On the forecasting of high-frequency financial time series based on arima model improved by deep learning," *Journal of Forecasting*, vol. 39, no. 7, pp. 1081–1097, 2020.

[91] H.-T. Yang, C.-M. Huang, and C.-L. Huang, "Identification of armax model for short term load forecasting: an evolutionary programming approach," in *Proceedings of Power Industry Computer Applications Conference*. IEEE, 1995, pp. 325–330.

[92] P. M. Maçaira, A. M. T. Thomé, F. L. C. Oliveira, and A. L. C. Ferrer, "Time series analysis with explanatory variables: A systematic literature review," *Environmental Modelling & Software*, vol. 107, pp. 199–209, 2018.

[93] A.-C. Petrică, S. Stancu, and A. Tindeche, "Limitation of arima models in financial and monetary economics." *Theoretical & Applied Economics*, vol. 23, no. 4, 2016.

[94] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[95] D. O. Hebb, *The organization of behavior: a neuropsychological theory.* J. Wiley; Chapman & Hall, 1949.

[96] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961.

[97] M. Tkáč and R. Verner, "Artificial neural networks in business: Two decades of research," *Applied Soft Computing*, vol. 38, pp. 788–804, 2016.

[98] R. L. Wilson and R. Sharda, "Bankruptcy prediction using neural networks," *Decision support systems*, vol. 11, no. 5, pp. 545–557, 1994.

[99] B. Back, G. Oosterom, K. Sere, and M. Van Wezel, "A comparative study of neural networks in bankruptcy prediction," in *Proc. Conf. on Artificial Intelligence Res. in Finland*, no. 12. Citeseer, 1994, pp. 140–148.

[100] G. Zhang, M. Y. Hu, B. E. Patuwo, and D. C. Indro, "Artificial neural networks in bankruptcy prediction: General framework and cross-validation analysis," *European journal of operational research*, vol. 116, no. 1, pp. 16–32, 1999.

[101] S. Lee and W. S. Choi, "A multi-industry bankruptcy prediction model using back-propagation neural network and multivariate discriminant analysis," *Expert Systems with Applications*, vol. 40, no. 8, pp. 2941–2946, 2013.

[102] Z. Zhao, S. Xu, B. H. Kang, M. M. J. Kabir, Y. Liu, and R. Wasinger, "Investigation and improvement of multi-layer perceptron neural networks for credit scoring," *Expert Systems with Applications*, vol. 42, no. 7, pp. 3508–3516, 2015.

[103] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[104] S. Urolagin, K. Prema, and N. S. Reddy, "Generalization capability of artificial neural network incorporated with pruning method," in *International Conference on Advanced Computing, Networking and Security*. Springer, 2011, pp. 171–178.

[105] M. Wooldridge, "Intelligent agents," *Multiagent systems*, vol. 6, 1999.

[106] J. Yen, A. Chung, H. Ho, B. Tam, R. Lau, M. Chua, and K. Hwang, "Collaborative and scalable financial analysis with multi-agent technology," in *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. 1999. HICSS-32. Abstracts and CD-ROM of Full Papers*. IEEE, 1999.

[107] O. Streltchenko, Y. Yesha, and T. Finin, "Multi-agent simulation of financial markets," *Formal Modelling in Electronic Commerce*, pp. 393–419, 2005.

[108] M. A. Souissi, K. Bensaid, and R. Ellaia, "Multi-agent modeling and simulation of a stock market," *Investment Management & Financial Innovations*, vol. 15, no. 4, p. 123, 2018.

[109] S. Ganesh, N. Vadori, M. Xu, H. Zheng, P. Reddy, and M. Veloso, "Multi-agent simulation for pricing and hedging in a dealer market," in *ICML'19 Workshop on AI in Finance*, 2019.

[110] K. Vora, S. Yagnik, and M. Scholar, "A survey on backpropagation algorithms for feedforward neural networks," *Int. J. Eng. Dev. Res*, vol. 1, no. 3, pp. 193–197, 2014.

[111] M. J. Bishop, "History and philosophy of neural networks," 2015.

[112] S.-L. Yan, Y. Wang, and J.-C. Liu, "Research on the comprehensive evaluation of business intelligence system based on bp neural network," *Systems Engineering Procedia*, vol. 4, pp. 275–281, 2012.

[113] P. K. Coats and L. F. Fant, "Recognizing financial distress patterns using a neural network tool," *Financial management*, pp. 142–155, 1993.

[114] E. Tölö, "Predicting systemic financial crises with recurrent neural networks," *Journal of Financial Stability*, vol. 49, p. 100746, 2020.

[115] Y. Jang, I. Jeong, and Y. K. Cho, "Business failure prediction of construction contractors using a lstm rnn with accounting, construction market, and macroeconomic variables," *Journal of management in engineering*, vol. 36, no. 2, p. 04019039, 2020.

[116] C. L. Cocianu and M. Avramescu, "Financial data forecasting using recurrent neural networks."

[117] Y. Hu, A. Huber, J. Anumula, and S.-C. Liu, "Overcoming the vanishing gradient problem in plain recurrent networks," *arXiv preprint arXiv:1801.06105*, 2018.

[118] M. Vochozka, J. Vrbka, and P. Suler, "Bankruptcy or success? the effective prediction of a company's financial development using lstm," *Sustainability*, vol. 12, no. 18, p. 7529, 2020.

[119] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[120] S. Edet, "Recurrent neural networks in forecasting s&p 500 index," *Available at SSRN 3001046*, 2017.

[121] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European journal of operational research*, vol. 270, no. 2, pp. 654–669, 2018.

[122] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PloS one*, vol. 12, no. 7, p. e0180944, 2017.

[123] Y. Liu, "Random forest algorithm in big data environment," *Computer Modelling & New Technologies*, vol. 18, no. 12A, pp. 147–151, 2014.

[124] W. Lin, Z. Wu, L. Lin, A. Wen, and J. Li, "An ensemble random forest algorithm for insurance big data analysis," *Ieee access*, vol. 5, pp. 16 568–16 575, 2017.

[125] Z. Rustam and G. S. Saragih, "Predicting bank financial failures using random forest," in *2018 International Workshop on Big Data and Information Security (IWBIS)*. IEEE, 2018, pp. 81–86.

[126] T.-H. Lee, A. Ullah, and R. Wang, "Bootstrap aggregating and random forest," in *Macroeconomic Forecasting in the Era of Big Data*. Springer, 2020, pp. 389–429.

[127] C. Liu, Y. Chan, S. H. Alam Kazmi, and H. Fu, "Financial fraud detection model: Based on random forest," *International journal of economics and finance*, vol. 7, no. 7, 2015.

[128] A. C. Müller and S. Guido, *Introduction to machine learning with Python: a guide for data scientists.* " O'Reilly Media, Inc.", 2016.

[129] W.-M. Lee, *Python machine learning.* John Wiley & Sons, 2019.

[130] S. Raschka, *Python machine learning.* Packt publishing ltd, 2015.

[131] D. Murray-Smith, *Modelling and Simulation of Integrated Systems in Engineering: Issues of Methodology, Quality, Testing and Application.* Woodhead Publishing, Limited, 2012.

[132] C. Teubert, M. J. Daigle, S. Sankararaman, K. Goebel, and J. Watkins, "A generic software architecture for prognostics (gsap)," *International journal of prognostics and health management*, vol. 8, no. 2, 2017.

[133] H. Mei, F. Chen, Y.-D. Feng, and J. Yang, "Abc: An architecture based, component oriented approach to software development," *Journal of Software*, vol. 14, no. 4, pp. 721–732, 2003.

[134] G. Gössler and J. Sifakis, "Composition for component-based modeling," *Science of Computer Programming*, vol. 55, no. 1-3, pp. 161–183, 2005.

[135] O. Ariyo, C. Eckert, and P. Clarkson, "Hierarchical decompositions for complex product representation," in *DS 48: Proceedings DESIGN 2008, the 10th International Design Conference, Dubrovnik, Croatia*, 2008.

[136] P. Lollini, A. Bondavalli, and F. Di Giandomenico, "A decomposition-based modeling framework for complex systems," *IEEE Transactions on Reliability*, vol. 58, no. 1, pp. 20–33, 2009.

[137] K. McInnis and S. Technologist, "Component-based development," *The concepts, technology and methodology. Castek Software Factory*, 2000.

[138] A. Sharma, R. Kumar, and P. Grover, "A critical survey of reusability aspects for component-based systems," *International Journal of Industrial and Manufacturing Engineering*, vol. 1, no. 9, pp. 420–424, 2007.

[139] C. Seceleanu and I. Crnkovic, "Component models for reasoning," *Computer*, vol. 46, no. 11, pp. 40–47, 2013.

[140] W. Goddard and S. Melville, *Research methodology: An introduction.* Juta and Company Ltd, 2004.

[141] U. Shafique and H. Qaiser, "A comparative study of data mining process models (kdd, crisp-dm and semma)," *International Journal of Innovation and Scientific Research*, vol. 12, no. 1, pp. 217–222, 2014.

[142] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI magazine*, vol. 17, no. 3, pp. 37–37, 1996.

[143] A. I. R. L. Azevedo and M. F. Santos, "Kdd, semma and crisp-dm: a parallel overview," *IADS-DM*, 2008.

[144] G. Mariscal, O. Marban, and C. Fernandez, "A survey of data mining and knowledge discovery process models and methodologies," *The Knowledge Engineering Review*, vol. 25, no. 2, pp. 137–166, 2010.

[145] R. M. Burton and B. Obel, "The validity of computational models in organization science: From model realism to purpose of the model," *Computational & Mathematical Organization Theory*, vol. 1, no. 1, pp. 57–71, 1995.

[146] F. P. Gibson, M. Fichman, and D. C. Plaut, "Learning in dynamic decision tasks: Computational model and empirical evidence," *Organizational Behavior and Human Decision Processes*, vol. 71, no. 1, pp. 1–35, 1997.

[147] D. Kaplan, M. Miller, and K. Carley, "Harmonization of computational models and experimental data: An illustration using data from the impact of wearable computers on cooperative work," *Social and Decision Sciences, Carnegie Mellon University, Pittsburgh, PA.*, 1996.

[148] C. M. Bishop, "Model-based machine learning," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1984, p. 20120222, 2013.

[149] M. Kulmanov, F. Z. Smaili, X. Gao, and R. Hoehndorf, "Machine learning with biomedical ontologies," *biorxiv*, 2020.

[150] ——, "Semantic similarity and machine learning with ontologies," *Briefings in bioinformatics*, vol. 22, no. 4, 2021.

[151] P. Lai, N. Phan, H. Hu, A. Badeti, D. Newman, and D. Dou, "Ontology-based interpretable machine learning for textual data," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–10.

[152] M. Austin, P. Delgoshaei, M. Coelho, and M. Heidarinejad, "Architecting smart city digital twins: Combined semantic model and machine learning approach," *Journal of Management in Engineering*, vol. 36, no. 4, p. 04020026, 2020.

[153] I. Jacobson, G. Booch, and J. Rumbaugh, "The unified software development process addison-wesley," *Reading, MA*, 1999.

[154] K. Vlaanderen, S. Jansen, S. Brinkkemper, and E. Jaspers, "The agile requirements refinery: Applying scrum principles to software product management," *Information and software technology*, vol. 53, no. 1, pp. 58–70, 2011.

[155] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," 2012.

[156] X. Wang and Y. He, "Learning from uncertainty for big data: future analytical challenges and strategies," *IEEE Systems, Man, and Cybernetics Magazine*, vol. 2, no. 2, pp. 26–31, 2016.

[157] X. Liu, N. Iftikhar, and X. Xie, "Survey of real-time processing systems for big data," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, 2014, pp. 356–361.

[158] S. Shrivastava and S. N. Pal, "Graph mining framework for finding and visualizing substructures using graph database," in *2009 International Conference on Advances in Social Network Analysis and Mining*. IEEE, 2009, pp. 379–380.

[159] A. Hogan, "Linked data & the semantic web standards." 2014.

[160] R. C. McColl, D. Ediger, J. Poovey, D. Campbell, and D. A. Bader, "A performance evaluation of open source graph databases," in *Proceedings of the first workshop on Parallel programming for analytics applications*, 2014, pp. 11–18.

[161] R. Myšková and P. Hájek, "Comprehensive assessment of firm financial performance using financial ratios and linguistic analysis of annual reports," *Journal of International Studies, volume 10, issue: 4*, 2017.

[162] J. L. Bellovary, D. E. Giacomino, and M. D. Akers, "A review of bankruptcy prediction studies: 1930 to present," *Journal of Financial education*, pp. 1–42, 2007.

[163] H. Wang and C. Kimble, "How external factors influence business model innovation: A study of the bosch group and the chinese automotive aftermarket," *Global Business and Organizational Excellence*, vol. 35, no. 6, pp. 53–64, 2016.

[164] D. Bidzhoyan and T. Bogdanova, "Modelling the financial stability of an enterprise taking into account macroeconomic indicators," *Business Informatics*, no. 3 (37), pp. 30–37, 2016. [Online]. Available: https://bijournal.hse.ru/en/2016--3(37)/195614303.html

[165] N. Yerashenia, A. Bolotov, G. Pierantoni, and D. Chan, "Semantic data pre-processing for machine learning based bankruptcy prediction computational model," in *2020 IEEE 22nd Conference on Business Informatics (CBI).* IEEE, 2020.

[166] D. L. Olson, D. Delen, and Y. Meng, "Comparative analysis of data mining methods for bankruptcy prediction," *Decision Support Systems*, vol. 52, no. 2, pp. 464–473, 2012.

[167] C. Junli and J. Licheng, "Classification mechanism of support vector machines," in *WCC 2000-ICSP 2000. 2000 5th International Conference on Signal Processing Proceedings. 16th World Computer Congress 2000*, vol. 3. IEEE, 2000, pp. 1556–1559.

[168] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry.* MIT press, 2017.

[169] S. E. Fahlman, "The cascade-correlation learning algorithm," *Advances in neural information processing systems*, vol. 2, pp. 525–532, 1990.

[170] D. Zhao, C. Huang, Y. Wei, F. Yu, M. Wang, and H. Chen, "An effective computational model for bankruptcy prediction using kernel extreme learning machine approach," *Computational Economics*, vol. 49, no. 2, pp. 325–341, 2017.

[171] Q. Yu, Y. Miche, E. Séverin, and A. Lendasse, "Bankruptcy prediction using extreme learning machine and financial expertise," *Neurocomputing*, vol. 128, pp. 296–302, 2014.

[172] D. D. Wu, Z. Yang, and L. Liang, "Using dea-neural network approach to evaluate branch efficiency of a large canadian bank," *Expert systems with applications*, vol. 31, no. 1, pp. 108–115, 2006.

[173] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[174] M. Leshno and Y. Spector, "Neural network prediction analysis: The bankruptcy case," *Neurocomputing*, vol. 10, no. 2, pp. 125–147, 1996.

[175] H. Adeli and S.-L. Hung, *Machine learning: neural networks, genetic algorithms, and fuzzy systems.* John Wiley & Sons, Inc., 1994.

[176] N. Yerashenia, A. Bolotov, and D. Chan, "Developing a generic predictive computational model using semantic data pre-processing with machine learning techniques and its application for stock market prediction purposes," in *2022 IEEE 24nd Conference on Business Informatics (CBI).* IEEE, 2022.

[177] A. Afify and H. E. Roman, "Estimating market index valuation from macroeconomic trends," *Quantitative Finance and Economics*, vol. 5, no. 2, pp. 287–310, 2021.

[178] D. Pilinkus *et al.*, "Macroeconomic indicators and their impact on stock market performance in the short and long run: the case of the baltic states," *Technological and Economic Development of Economy*, no. 2, pp. 291–304, 2010.

[179] B. Weng, W. Martinez, Y.-T. Tsai, C. Li, L. Lu, J. R. Barth, and F. M. Megahed, "Macroeconomic indicators alone can predict the monthly closing price of major us indices: Insights from artificial intelligence, time-series analysis and hybrid models," *Applied Soft Computing*, vol. 71, pp. 685–697, 2018.

[180] D. Shah, H. Isah, and F. Zulkernine, "Stock market analysis: A review and taxonomy of prediction techniques," *International Journal of Financial Studies*, vol. 7, no. 2, p. 26, 2019.

Appendicies

A A Brief Manual on Creating Graphs in Neo4j Environment

To get started with Neo4j, a new local database should be created. After launching it, the link can be used *HTTP://localhost:7474/browser/* or the Neo4j Browser. At the top of the Neo4j Browser window is a line of the so-called main editor.

Starting a set of commands with a colon – ':', a list of all available commands with a short description will appear.

The main elements Cypher language operates on are the graph's vertices (nodes) and edges.

Edges in Neo4j have type and direction, vertices can be marked with one or more labels, and they can also have several additional properties. Cypher uses ASCII graphics to describe the patterns of the extracted information.

Vertices are written in the form of a pair of parentheses, inside which additional conditions are set, be it the type of the vertex or the value of some property, for example, *(n: Ratio)* - the vertex of the Ratio type. Edges are written as a pair of square brackets; by analogy with the vertices inside the brackets, additional conditions are set. *[m: subClassOf]* - relationship of the subClassOf type.

The direction of the link is set using the arrows as follows:

```
() - [] -\> (),
() \<- [] - () or
() - [] - () for the case when the direction of the link is not important
```

In the case when there is no need to impose additional restrictions on the connections between the vertices, we can omit the square brackets in the connection indication:

```
() - ()
```

The main key commands for creating and retrieving information are: CREATE is a keyword that creates nodes or links between them; MATCH is a keyword followed by a pattern describing the information we are looking for; WHERE is a keyword after which we can add additional restrictions imposed on the template or filter the results; RETURN is a keyword that defines what will be returned as a result of a request.

So, for example, the following code will create all the nodes, the elements to which it is related, plus the connections between them, and will display this graph as a result. The example was based on one of the Financial Ratios used as an input for *Use Case 1* modification of the generic model (see Section 4)

```
Create (rf1:Ratio {ratioID:"Gearing", normative_value:"1", linguistic_variable:
"Financial Sustainability", value: 31.16429818, weight: 0.5771})
Create (b8:Criteria {criteriaId:"Longterm Liabilities", year:"2018",
type:"Liabilities"})
Create (b9:Criteria {criteriaId:"Shareholders' Funds", year:"2018",
type:"Liabilities"})
Create (b10:Criteria {criteriaId: "Short Term Loans and Overdrafts", year:"2018",

Create (rf1)-[:directly_related_to]->(b8)
Create (rf1)-[:directly_related_to]->(b10)
Create (rf1)-[: inversely_related_to]->(b9)
MATCH (n) RETURN n;
```

B Ontology of Bankruptcy Prediction (Full Version)

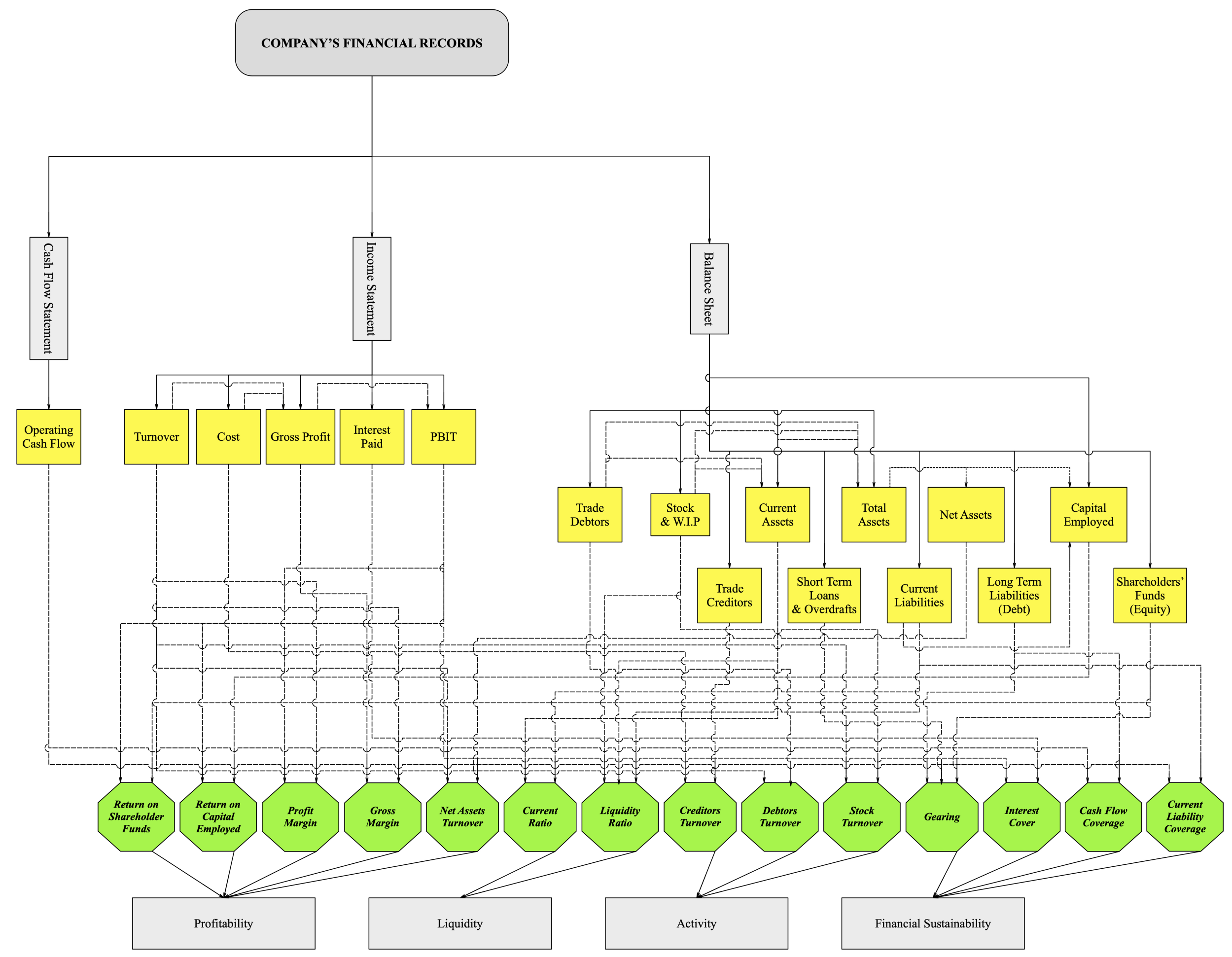
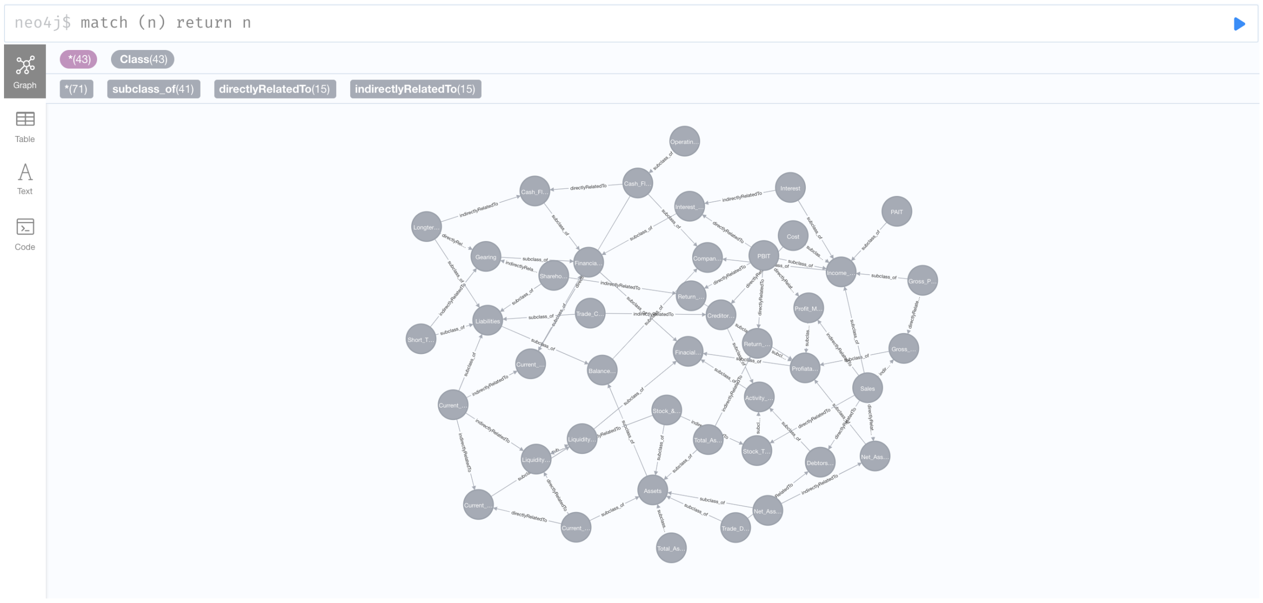


Figure 24: A complete diagram of Ontology of Bankruptcy Prediction

C Ontology of Bankruptcy Prediction in Neo4j Environment (Full Version)



```
neo4j$ match (n) return n
```

Figure 25: Ontology of Bankruptcy Prediction presented in Neo4j

D Ontology of Market Index Prediction in Protege Environment: Hierarchical Connections Only

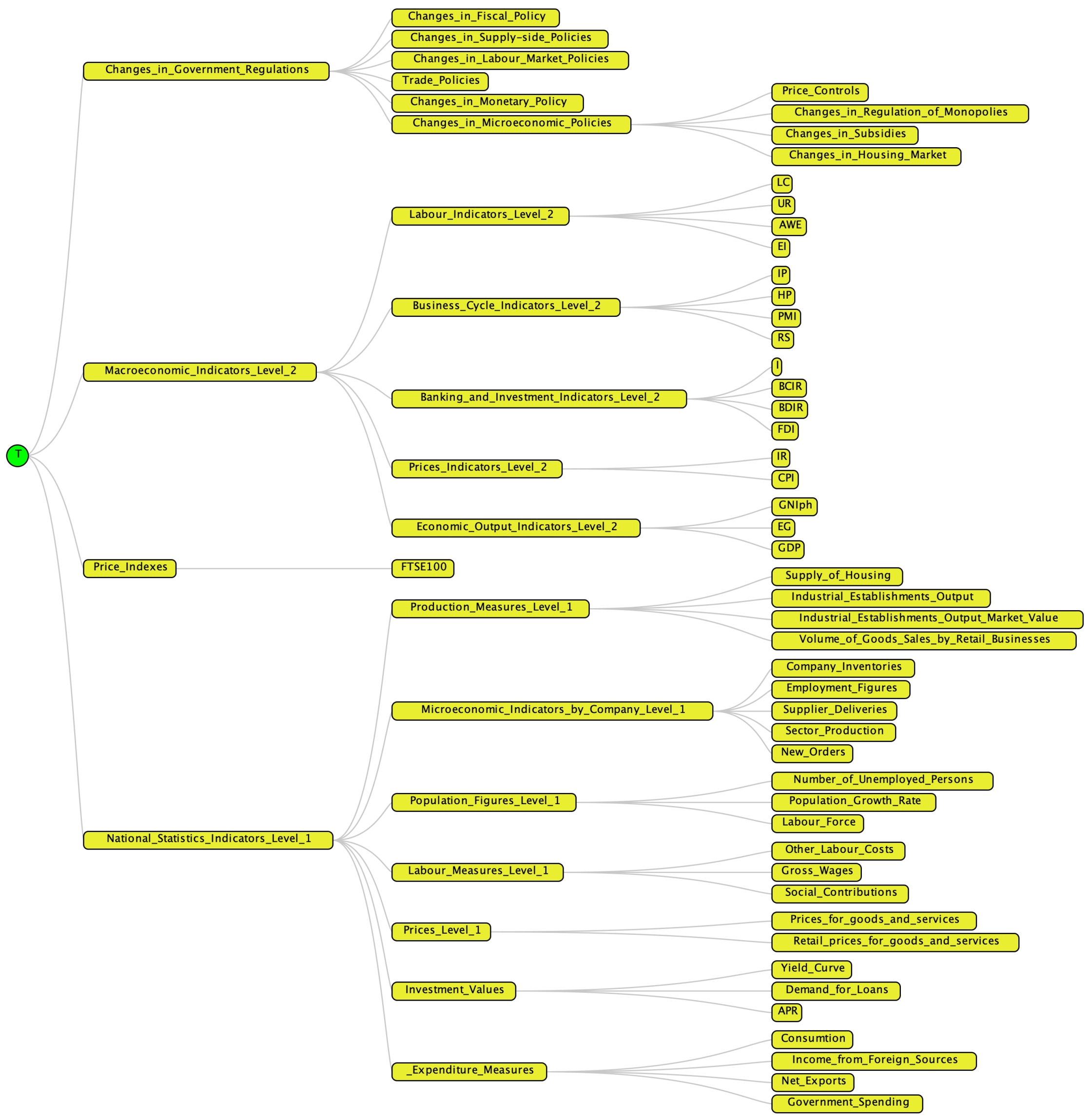
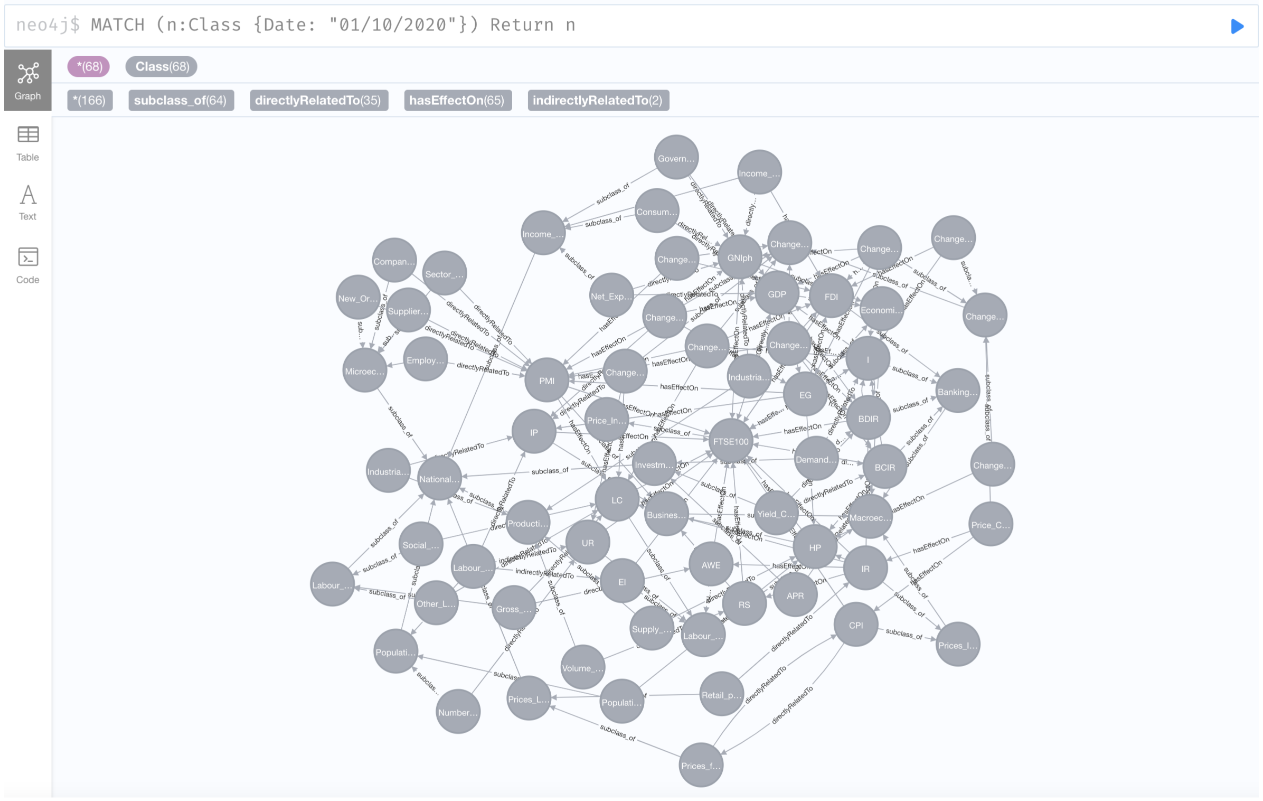

T
Changes_in_Government_Regulations
Changes_in_Fiscal_Policy
Changes_in_Supply-side_Policies
Changes_in_Labour_Market_Policies
Trade_Policies
Changes_in_Monetary_Policy
Changes_in_Microeconomic_Policies
Price_Controls
Changes_in_Regulation_of_Monopolies
Changes_in_Subsidies
Changes_in_Housing_Market
Macroeconomic_Indicators_Level_2
Labour_Indicators_Level_2
LC
UR
AWE
EI
Business_Cycle_Indicators_Level_2
IP
HP
PMI
RS
Banking_and_Investment_Indicators_Level_2
I
BCIR
BDIR
FDI
Prices_Indicators_Level_2
IR
CPI
Economic_Output_Indicators_Level_2
GNIph
EG
GDP
Price_Indexes
FTSE100
National_Statistics_Indicators_Level_1
Production_Measures_Level_1
Supply_of_Housing
Industrial_Establishments_Output
Industrial_Establishments_Output_Market_Value
Volume_of_Goods_Sales_by_Retail_Businesses
Microeconomic_Indicators_by_Company_Level_1
Company_Inventories
Employment_Figures
Supplier_Deliveries
Sector_Production
New_Orders
Population_Figures_Level_1
Number_of_Unemployed_Persons
Population_Growth_Rate
Labour_Force
Labour_Measures_Level_1
Other_Labour_Costs
Gross_Wages
Social_Contributions
Prices_Level_1
Prices_for_goods_and_services
Retail_prices_for_goods_and_services
Investment_Values
Yield_Curve
Demand_for_Loans
APR
_Expenditure_Measures
Consumtion
Income_from_Foreign_Sources
Net_Exports
Government_Spending

Figure 26: Ontology of Market Index Prediction presented in Protege (SOVA plugin visualisation)

E Ontology of Market Index Prediction in Neo4j Environment (Full Version, the Latest Time-Slot)



```
neo4j$ MATCH (n:Class {Date: "01/10/2020"}) Return n
```

Figure 27: Ontology of Market Index Prediction presented in Neo4j

F The Architecture of a Hybrid Machine Learning Engine for PIPCM (Full Version)

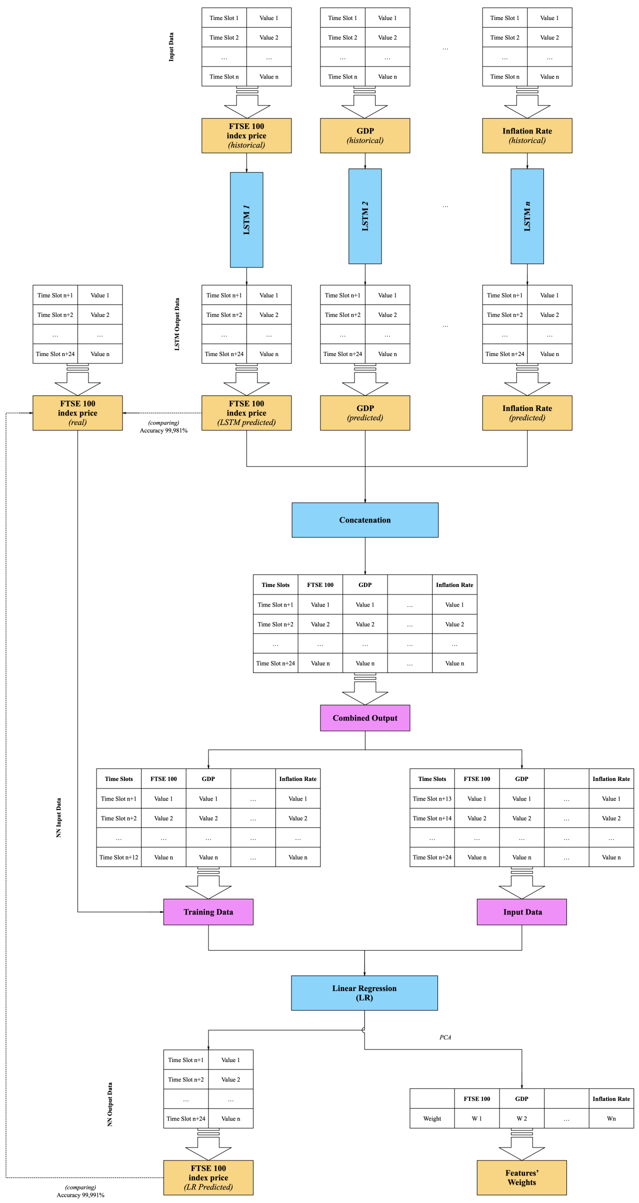


Figure 28: PIPCM Hybrid MLE Architecture

G The Results of Macroeconomic Indictors Predictions using LSTMs

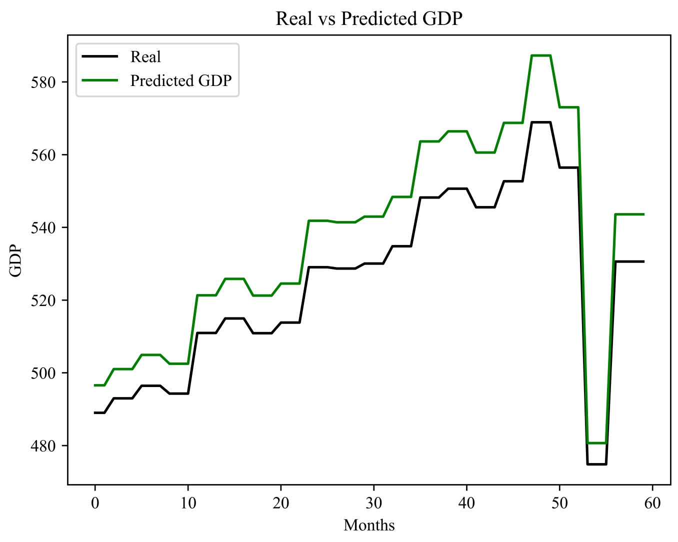


Figure 29: Macroeconomic Indicators LSTM Prediction: GDP

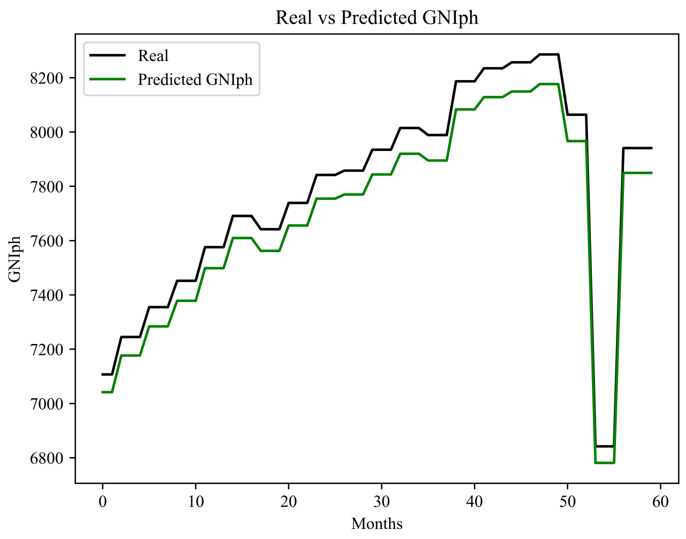


Figure 30: Macroeconomic Indicators LSTM Prediction: GNIph

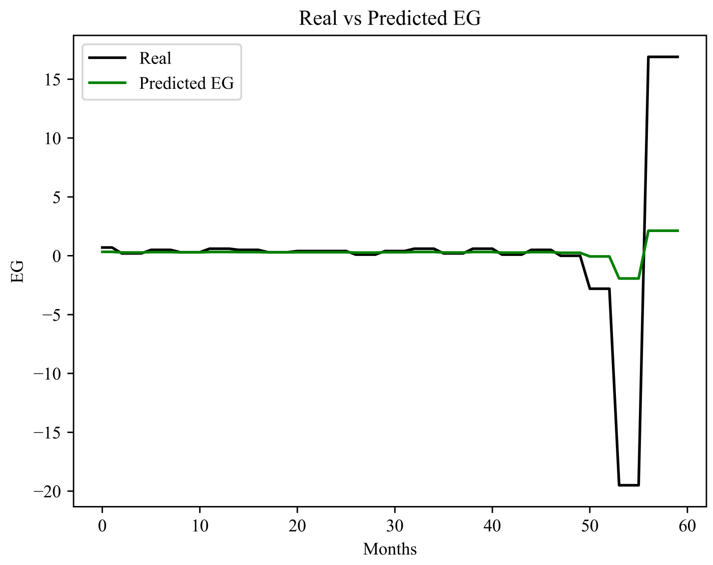


Figure 31: Macroeconomic Indicators LSTM Prediction: EG

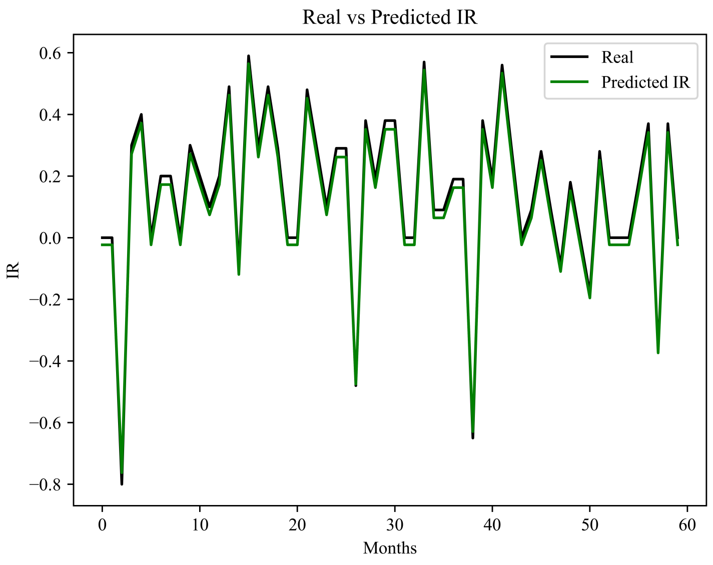


Figure 32: Macroeconomic Indicators LSTM Prediction: IR

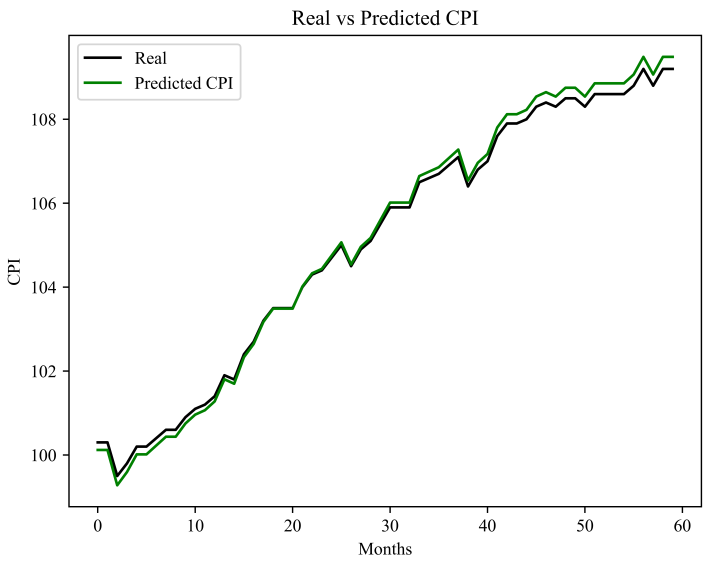


Figure 33: Macroeconomic Indicators LSTM Prediction: CPI

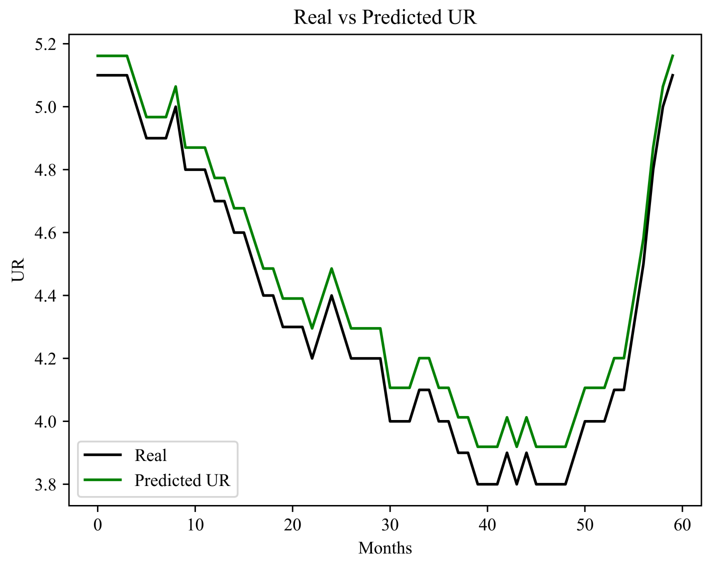


Figure 34: Macroeconomic Indicators LSTM Prediction: UR

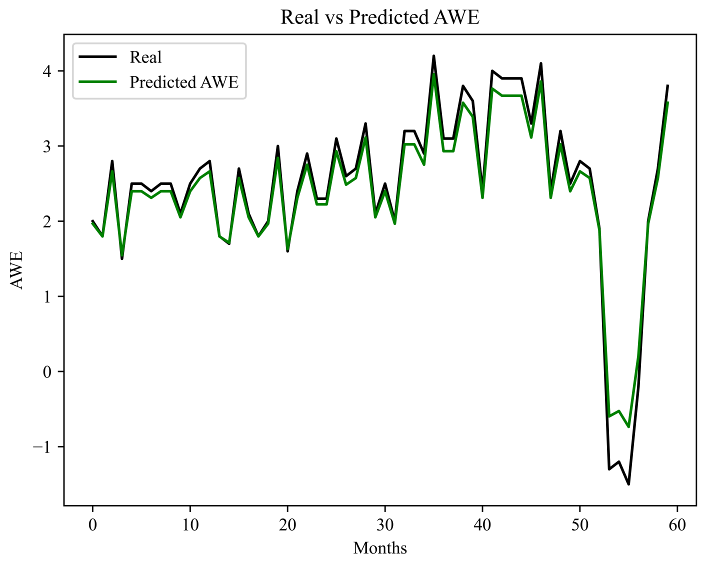


Figure 35: Macroeconomic Indicators LSTM Prediction: AWE

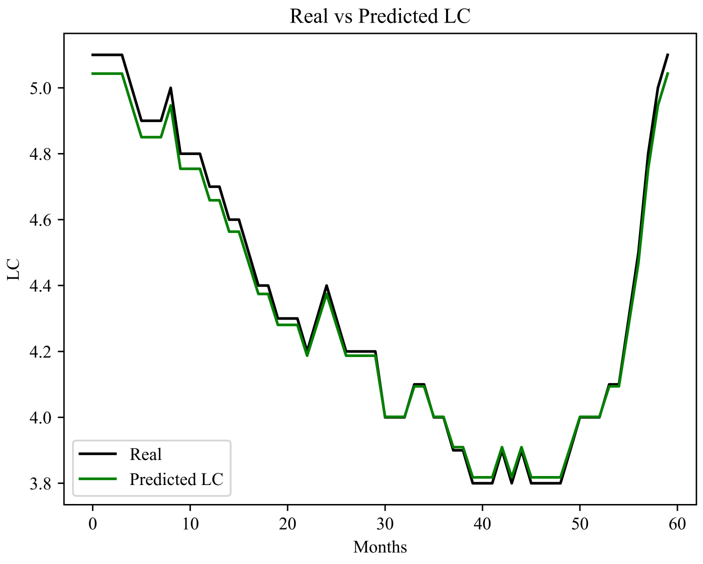


Figure 36: Macroeconomic Indicators LSTM Prediction: LC

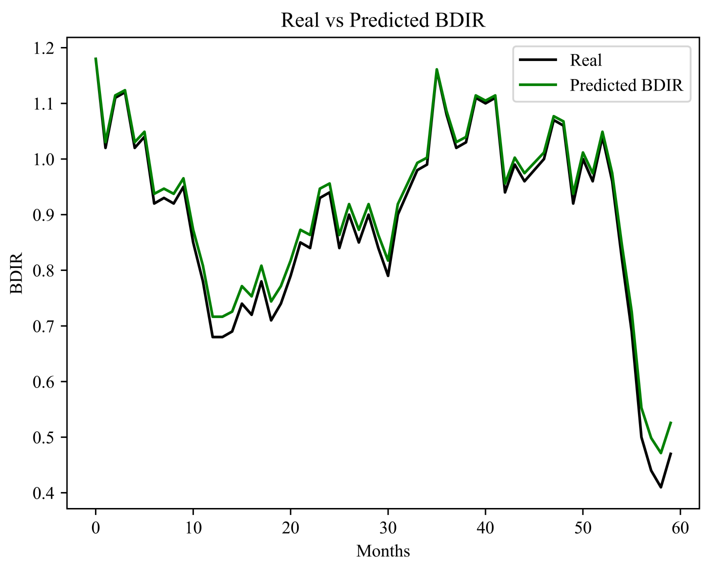


Figure 37: Macroeconomic Indicators LSTM Prediction: BDIR

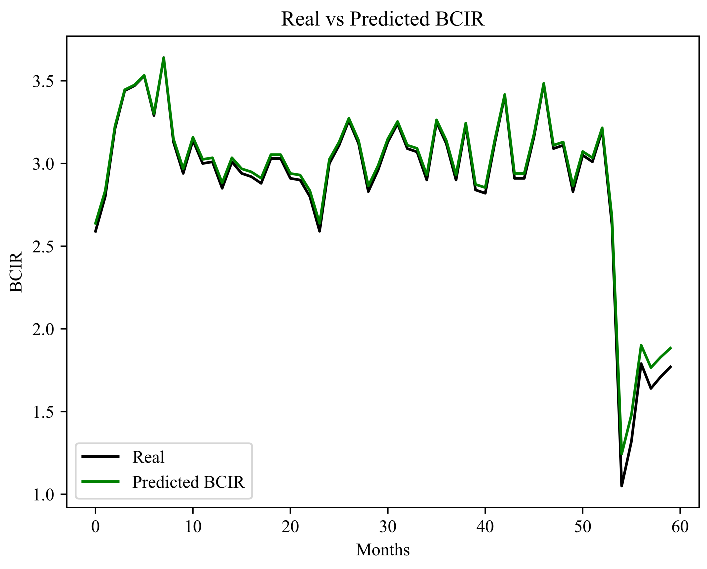


Figure 38: Macroeconomic Indicators LSTM Prediction: BCIR

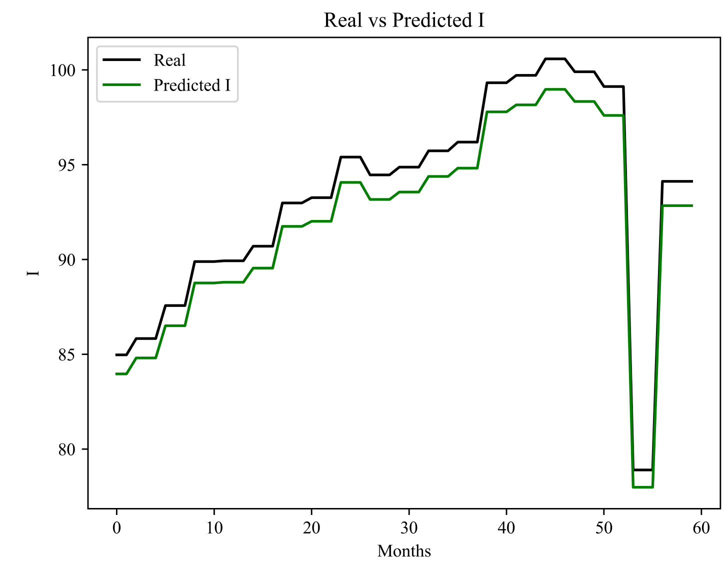


Figure 39: Macroeconomic Indicators LSTM Prediction: I

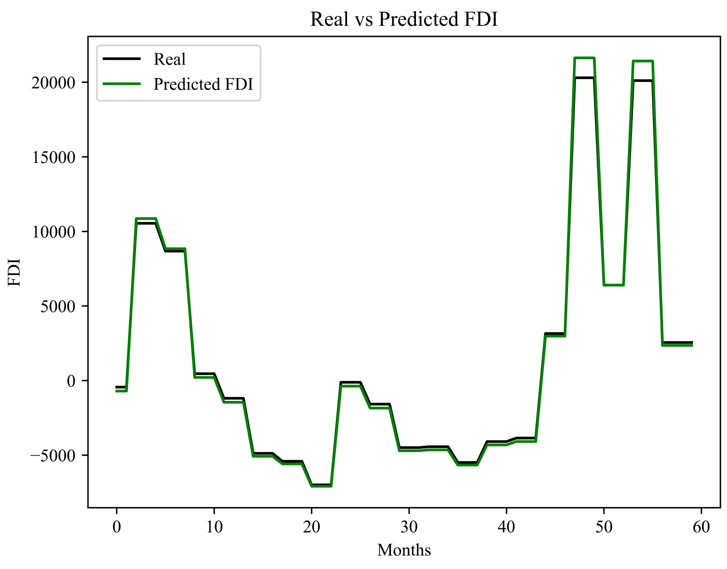


Figure 40: Macroeconomic Indicators LSTM Prediction: FDI

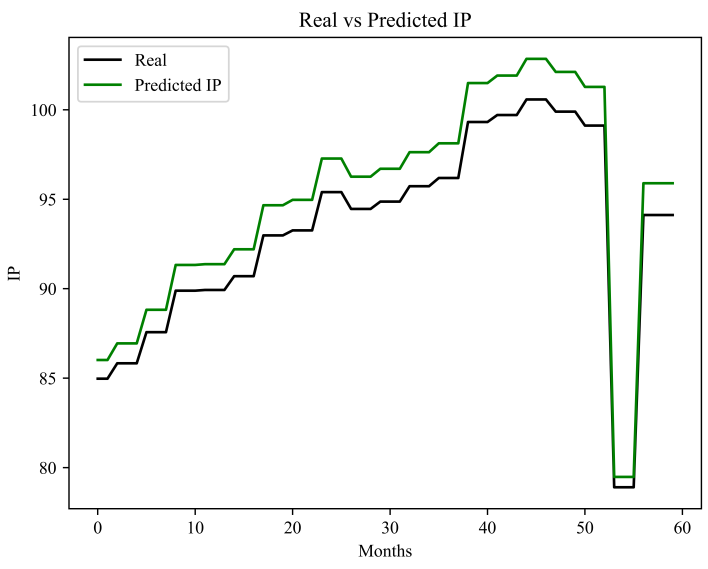


Figure 41: Macroeconomic Indicators LSTM Prediction: IP

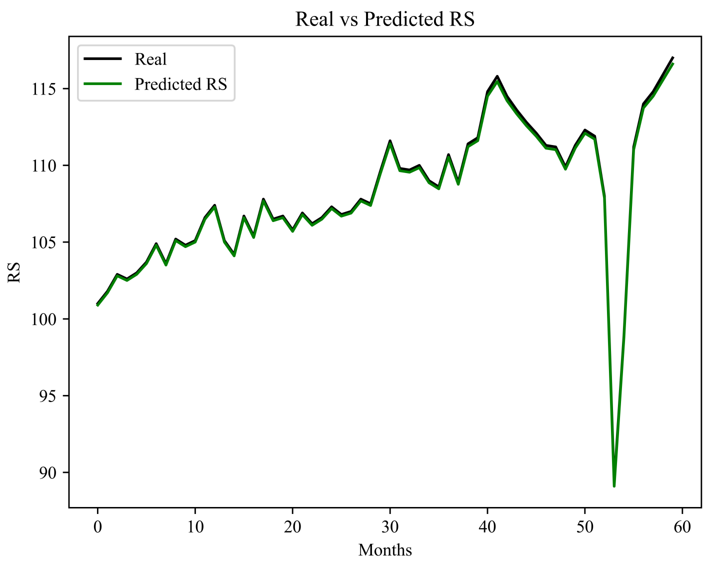


Figure 42: Macroeconomic Indicators LSTM Prediction: RS

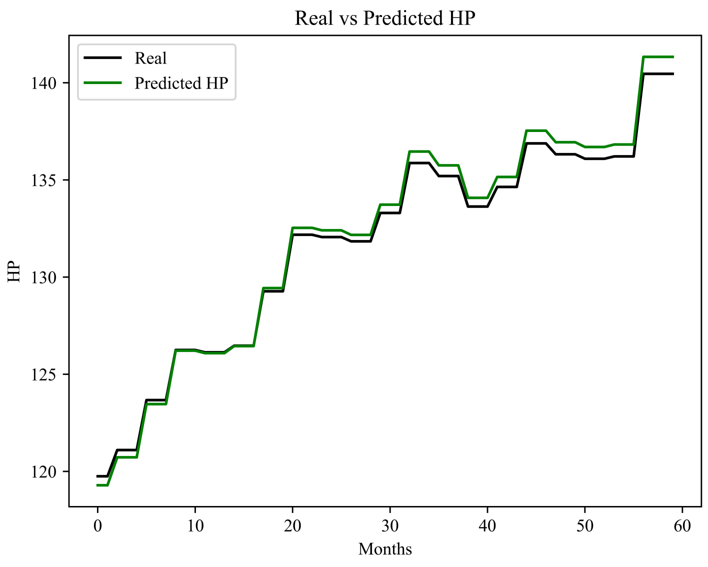


Figure 43: Macroeconomic Indicators LSTM Prediction: HP

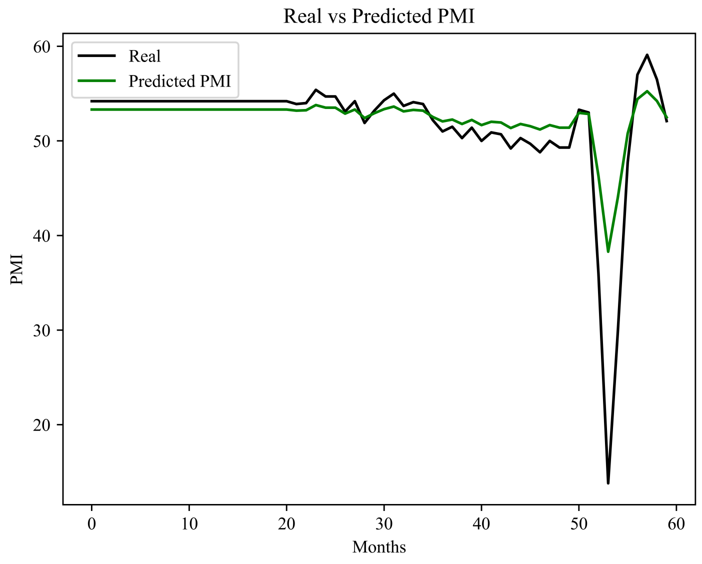


Figure 44: Macroeconomic Indicators LSTM Prediction: PMI

H List of Files Attached

All the files are available on my GitHub: `https://github.com/Yerashenia/Predictive-Computational-Model-PCM`

Use Case 1

- *weights_importance_uc1.csv* – CSV file containing Random Forest Feature Importance Prediction Results.
- *BPCM_NN_v.2.0.py* – Python Code of BPCM Neural Network.
- *BPCM_NN_testing.py* – The Python Code for Neural Network evaluation purposes.
- *Company_A_Fin_Indicators.csv* – CSV file containing Company A data (collected from the FAME database).
- *Company_A_Fin_Ratios.csv* – CSV file containing Company A data (collected from the FAME database).
- *cypher_queries_uc1.py* – Python Code of Feature Selection in Neo4j.
- *feature_importance_uc1.py* – Python Code of Random Forest Feature Importance Prediction.
- *import_to_neo4j_uc1.py* – Python Code: Importing the OBP Ontology to Neo4j and Filling the Graph Database with data.
- *Input_Data_with_Neo4j_Feature_Selection.csv* – CSV file with Selected Data imported from Neo4j.
- *NN_with_neo4j_feature_selection.py* – Python Code of BPCM Neural Network with Feature Selection.
- *OBP_Ontology_v.2.1.owl* – OBP Ontology OWL file.
- *Ratios_Export.csv* – CSV file containing Input Data imported from Neo4j.
- *Training_Data_2017_71.csv* – CSV file containing Complete Dataset for BPCM Neural Network for 2017.

- *Training_Data_2018_86.csv* – CSV file containing Complete Dataset for BPCM Neural Network for 2018.
- *Training_Data_2019_87.csv* – CSV file containing Complete Dataset for BPCM Neural Network for 2019.
- *Training_Data_2020_94.csv* – CSV file containing Complete Dataset for BPCM Neural Network for 2020.
- *Training_Data_2021_113.csv* – CSV file containing Complete Dataset for BPCM Neural Network for 2021.
- *training_data_file_after_neo4j_fs.py* – Python Code: Data file after Feature Selection generation.
- *Training_Data_with_Neo4j_Feature_Selection.csv* – CSV file containing Data after Feature Selection for BPCM Neural Network.
- *BPCM_NN_cross-validation_testing.py* – Python Code: BPCM Neural Network Cross-Validation.
- *BPCM_LogReg_testing.py* – Python Code of Logistic Regression model applied to BPCM datasets.

Use Case 2

- *weights_importance_uc2.csv* – CSV file containing Random Forest Feature Importance Prediction Results.
- *combined_output.csv* – CSV file containing the combined output of LSTMs generated in the Concatenation Unit. *cypher_queries_uc2.py* – Python Code containing Feature Selection in Neo4j.
- *feature_importance_uc2.py* – Python Code of Random Forest Feature Importance Prediction.
- *feature_selection_from_neo4j.csv* – CSV file containing Input Data after Feature Selection imported from Neo4j.
- *fill_ontology_in_neo4j_uc2.py* – Python Code: Filling the Neo4j database with Data.
- *final_results_feature_selection_neo4j.csv* – CSV file containing Linear Regression final results with Feature Selection.
- *final_results.csv* – CSV file containing Linear Regression final results (without Feature Selection).
- *import_to_neo4j_uc2.py* – Python Code: Importing the OMIP Ontology to Neo4j.
- *Input_Data_from_Neo4j.csv* – CSV file containing Data imported from Neo4j.
- *input_data_import_from_neo4j.py* – Python Code: Importing Input Data from Neo4j.

- *linear_regression_feature_selection_neo4j.py* – Python Code of MIPCM Linear Regression with Feature Selection.
- *linear_regression.py* - Python Code of MIPCM Linear Regression (without Feature Selection).
- *main_lstms_concatenation_unit.py* – Python Code: Concatenation Unit.
- *Price_Index_Prediction_v3.2.owl* – OMIP Ontology RDF file (saved as OWL).
- *Processed_Input_Data_FTSE100_1985_21.csv* – CSV file containing Input Data taken from external sources.
- *lstm_AWE.py, ..., lstm_UR.py* – Python Code of single LSTMs for MIPCM.
- *ARIMAX_FTSE100.py* – Python Code of ARIMAX model to predict FTSE100 close prices.