Modelling and Mining of Data mining Workflows in Data Science Scripts

Thesis
Submitted in the partial fulfilment of the requirement for the degree of MASTER OF SCIENCE in COMPUTER SCIENCE

By
Jan Zegers

Web Information System
Department of Computer Science, EEMCS, Delft University of Technology, Delft, the Netherlands
Born in Hoorn, Netherlands
Graduation Committee

Chair: prof. dr. ir. G-J. Houben, Faculty EEMCS, TU Delft.
University supervisor: dr. C. Lofi, Faculty EEMCS, TU Delft.
Committee Member: dr.ir. J.A., Pouwelse, Faculty EEMCS, TU Delft.
Committee Member: dr. A. Bozzon, Faculty EEMCS, TU Delft.

Copyright © 2017 by J.G. Zegers
An electronic version of this dissertation is available at http://repository.tudelft.nl/.
Abstract

Knowledge discovery and data mining (KDD) is the process that extracts new knowledge from data. Data scientists program their solutions of their KDD challenges in data mining programming scripts. Data mining programming scripts contain multiple or all steps of the KDD process. In other words, these scripts start with data and end with knowledge. Learning from the knowledge and experience documented in the scripts would be beneficial, but till now a formalized sharing of this information between Data Scientists is missing. This is partly because there is no formalized model to share it and partly due to intellectual property of companies on the solution of their employed Data Scientist. In this thesis, we describe an open source repository of data mining scripts in order to find out how data mining solutions are constructed. Gaining knowledge on these solutions and making it possible to share this knowledge.

The scripts are described by an adapted KDD process model. This model extends the existing models, while these existing models were too conceptual and abstract to describe and classify data mining scripts. The extension of the current models consists of an additional layer of Methods that connects the abstract models and the data mining scripts. This additional layer has been constructed by combining various KDD ontologies.

After the modelling, the script lines are labelled with the methods from the adapted KDD process model. This is achieved by going through the different steps of a KDD process. During the transformation from text to numbers, required for training an automatic classifier, word embedding techniques are applied. Hereby, the implemented Word2Vec algorithm learns what the semantically similar words are within the code. Script lines with semantically similar words get a similar vector. This makes it possible to train a supervised classifier that assigns labels to script lines. This results in a list of labels per script describing the KDD process of the script.

These lists of labels have been analysed in various ways. First, it can be demonstrated that data scientists usually use two or more script lines to perform a Step or Task. Furthermore, we analyse which Methods are often used together. This makes it possible to recommend appropriate Methods during a future process. Lastly, we studied the transitions in the scripts from Step to Step, Task to Task and Method to Method, of the adapted KDD model. Here we see that the conceptual process model is indeed reflected in the scripts. In addition, we see that the scripts, like the process, are cyclic and iterative, often returning to an earlier step. Lastly commonly used transitions in the script are visualized and describes.

The description of scripts gives insight into the choices Data Scientist make to solve their KDD challenges. This is useful information if you want to contribute towards solving newly encountered KDD challenges. Furthermore, the extended model and the developed workflow offer starting points for future work that supports the knowledge discovery and data mining process.
# Contents

Abstract .................................................................................................................. ii

Contents .................................................................................................................. iii

List of Figures ........................................................................................................ v

List of Tables ......................................................................................................... v

1 Introduction ......................................................................................................... 1

1.1 Research Questions ......................................................................................... 3

1.2 Method ............................................................................................................. 3

1.3 Contribution ..................................................................................................... 6

1.4 Thesis Outline .................................................................................................. 6

2 Theory of Knowledge Discovery and Data mining ............................................. 7

2.1 Survey of existing process models and ontologies ......................................... 7

2.1.1 Process Models .......................................................................................... 7

2.1.2 KDD Ontologies ....................................................................................... 10

2.2 Evaluation of process models and ontologies ................................................. 14

3 Adapted KDD model .......................................................................................... 15

3.1 Model requirements ....................................................................................... 15

3.2 Structure and Sources ................................................................................... 16

3.3 Model design rules ......................................................................................... 17

3.4 Developed model ........................................................................................... 21

3.5 Conclusion ..................................................................................................... 22

4 Classification of scripts ..................................................................................... 24

4.1 Selection of Target Data ............................................................................... 24

4.2 Pre-processing ............................................................................................... 25

4.3 Transformation by feature extraction ............................................................ 25

4.3.1 Word-embedding ...................................................................................... 26

4.3.2 Word2Vec .................................................................................................. 26

4.3.3 Doc2Vec ..................................................................................................... 27

4.3.4 Py2Vec ....................................................................................................... 27

4.3.5 Building of the Word2Vec models ............................................................ 27

4.4 Data mining ................................................................................................... 28

4.4.1 Unsupervised Classification .................................................................... 28

4.4.2 Supervised Classification ......................................................................... 29

4.4.3 Evaluation ................................................................................................ 30

4.5 Classification of all the script lines ............................................................... 31

4.5.1 Numerical Analysis .................................................................................. 31
4.6 Conclusion........................................................................................................................................33
5 Patterns and Insights in Data mining scripts.........................................................................................35
  4.1 Code Blocks........................................................................................................................................35
  4.2 Co-occurrence.....................................................................................................................................37
  4.3 Transition state analysis....................................................................................................................39
  4.4 Conclusion........................................................................................................................................45
6 Discussion and Future work..................................................................................................................48
7 Conclusion and Outlook..........................................................................................................................53
  7.1 Conclusion........................................................................................................................................53
  7.2 Outlook ............................................................................................................................................54
Acknowledgment.......................................................................................................................................55
Bibliography..............................................................................................................................................56
List of Figures

Figure 1-1: Schema, connection abstract and specific layers of the KDD process ........................................ 2
Figure 1-2: Overview of method ................................................................................................................. 5
Figure 2-1 Evolution of Data mining models and methodologies ...................................................................... 7
Figure 2-2 Outlined process model of Fayad .............................................................................................. 8
Figure 2-3 Four level breakdown of the CRISP-DM methodology ............................................................. 9
Figure 2-4 Phases of the CRISP-DM reference model .................................................................................. 10
Figure 2-5 The core concepts of DMOP .................................................................................................... 11
Figure 2-6 Core DM task hierarchy from the DMOP model ......................................................................... 12
Figure 2-7 KDDONTO main classes and relationships .............................................................................. 13
Figure 2-8 Representation of a data mining task in OntoDM .................................................................. 14
Figure 3-1: Overview of conceptual model structure .................................................................................. 17
Figure 3-2: Barplot of occurrences of top ten libraries in Kaggle script .................................................. 17
Figure 3-3: Model Design Steps .............................................................................................................. 18
Figure 3-4: The adapted process model .................................................................................................... 20
Figure 4-1: An example of the filtering of the line of code ........................................................................ 28
Figure 4-2: Distribution of the labels over the Step classes ...................................................................... 32
Figure 4-3: The distribution of the labels: .................................................................................................. 33
Figure 5-1: Histogram of blocklength of different filter levels for Step Level. ......................................... 36
Figure 5-2: Histogram of Block length of the labels over the different model levels .............................. 36
Figure 5-3: Co-occurrence on Step level .................................................................................................. 37
Figure 5-4: Co-occurrence on Task level .................................................................................................. 38
Figure 5-5: Co-occurrence on Method Level ............................................................................................ 39
Figure 5-6: State transition on Step level, without filters ........................................................................ 40
Figure 5-7: State transition on Step level, without self-loops ................................................................. 41
Figure 5-8: Transition state diagram on Step level without Pre-processing ........................................... 42
Figure 5-9: State transition on Step level, filtering out pre-processing .................................................... 43
Figure 5-10 State transition on Task level ............................................................................................... 44
Figure 5-11 State transition on Method level .......................................................................................... 45

List of Tables

Table 3-1: The adapted KDD process model ............................................................................................ 23
Table 4-1: Scraped scripts split up by programming language .................................................................... 24
Table 4-2: Examples of labelled lines of Code ......................................................................................... 25
Table 4-3: Average result of the classification with Word2Vec models .................................................. 30
Table 4-4: Results of classification of full dataset .................................................................................... 31
Table 4-5: Confusion matrix of the classified labels on Step level ............................................................ 32
Table 4-6: Mapping of the (mis-) classified labels with the Neural Network on Method level ............... 34
Table 4-1: Transition matrix, from one Step to another Step, with self-loops ........................................ 41
Table 4-2: State transition on Step level, without self-loops .................................................................. 42
Table 4-3: Transition matrix, from one Task to another Task ................................................................ 46
Table 4-4: Transition matrix, from one Method to another Method ...................................................... 47
1 Introduction

Big data has become one of the most valuable resources in modern business and it has been predicted to be the biggest game changer for society in the next ten years (Bestuursstaf, 2011; Fenema van & Et.al., 2015). Especially for corporations and governments, big data is expected to offer a multitude of benefits because these entities - by their very nature - already gather detailed information on their performance, their clients, and the market. Big data can be exploited to systematically reveal information and discover trends that are either explicitly or implicitly contained in this wealth of information. This information is expected to be of value for the organisation, for example improving the effectiveness of its operations or leading to a better understand their clients and the market in which they operate.

In the corporate and governmental context, Data Scientists act as miners that use multiple techniques to extract knowledge from the data. Often operating in online communities, Data Scientists have a large toolbox of methods and algorithms at their disposal for mining data. This large toolbox is constructed by the continuous effort on research and development on better techniques to extract knowledge from data. However, with all these techniques at hand, it is not always obvious how to construct a data mining workflow and to come from data to knowledge.

Large groups of Data Scientists are working on the same Knowledge Extraction problems, hence it would be beneficial to learn from previous experiences. So far, there is a lack of [systematic and] formalized sharing of knowledge between Data Scientists about how the problems are solved especially with respect to data mining workflows. As a result, ad hoc methods such as searching for individual solutions on the internet are not uncommon. An additional problem is that in the commercial part of the data science community, companies keep their solutions as part of their intellectual property and value, or solutions are part of closed ecosystems, such as Workflow Management Systems. It is clear that the ad-hoc and opaque approach to find a suitable data mining workflow that starts with data and ends new Knowledge, is inefficient and wanting.

The need for formalized knowledge sharing was identified by other and as a result, public sources of data mining workflow solutions are appearing. Typically, these solutions are published online on data-scientist community platforms as submissions of data mining challenges or as showcases how to approach a dataset. The biggest platform to host this kind of challenges is Kaggle\(^1\). Kaggle started as a website which hosted competitions between Data Scientist, with prices for the best predictive solution. Nowadays it is evolved in a much broader platform. For example, Kaggle incorporates multiple data mining tutorials. Furthermore, it is a place where Data Scientist can work together on exploration of uploaded datasets with social impact. With all the challenges and projects together, Kaggle holds more than 30 thousand uploaded challenge-solutions from which (in principle) a data mining workflow solution could be extracted.

The need of knowledge sharing has also been addressed from a more fundamental point of view. Previous work on data mining workflows focusses on the description of the conceptual data mining process, which is called the Knowledge Discovery and Data mining process (KDD). The first effort made to formalize the process is the KDD process model of Fayyad (Fayyad, Piatetsky-shapiro, & Smyth, 1996). Fayyad made a science-oriented model of the KDD process; build by questioning data-analysis experts with a scientific or industrial background. Another important model is the CRISP-DM model (Chapman et al., 2000), which is built as the industrial standard to discuss and manage the

\(^1\) www.kaggle.com
process. Both conceptual models focus on managing and discussing the KDD process, resulting in a high-level description of the process.

**Figure 1-1: Schema, connection abstract and specific layers of the KDD process**

The top level KDD process models divide the process into phases or steps. All models split up the steps in smaller sub-units which are here called tasks. Tasks are actionable items that you can decide to execute accomplish the part of the step. The definition of these tasks can be implicit or explicit depending on the model. In addition to the abstract top levels models, there is a notion of the more specific methods and implementations layers. However, existing KDD process models refrain from specifying these more specific Method and Implementation layers because they want to stay independent and generally applicable. However, here a method can be thought of as a specific methodology taken to accomplish a task (e.g. a logistic regression classifier and a KNN classifier both accomplish the task of classification). Each method can be implemented using 1 or more specific implementations relying on a specific tool set (a specific library implementing the logistic regression algorithm). Figure 1-1 shows these abstract and specific layers and their mutual inter-dependence in one schema. The limitation of these top-level KDD process models, such as Fayyad’s and CRISP DM, is that they only define the abstract layers. This makes them not directly useful when studying the implementation layers.

In contrast to the aforementioned research, in this work, I studied the specific Method and Implementation layers. I studied how to automatically analyse data mining programming scripts in order to classify them into the Step, Task and Methods of the KDD process. This classification could be used to automatically generate a description of the process. For this purpose, I extended the KDD process model into an adapted KDD process model which breaks down the abstract steps in smaller task and methods. This adapted model is more specific and therefore it enables richer ways to express the KDD process.

With the adapted KDD process model I classified a set of Kaggle scripts. This lead to a repository of annotated scripts, each representing an individual workflow. The analysis of these workflows gives insight how Data-Scientists solve the data mining challenges. This knowledge can be used in the future as assistance in new challenges. For example, this can take the form of a plugin for programming applications aiding the user in selecting an appropriate method for a task. Moreover, this work is one of the first step towards a more model driven approach to workflow design.
1.1 Research Questions
Several research questions have been defined to guide this research; the main research question of this work is defined as:

- **RQ: To what extent can data mining programming scripts be described and classified by a KDD process model?**

The scope of this work is limited to analysing Python scripts published on Kaggle. A KDD process model in this project is a conceptual model, describing the process of Knowledge Discovery and Data mining. Such a conceptual model makes it possible to discuss and reason about the process and the content of the scripts. Linking the scripts to a conceptual model should make it possible to extract the knowledge put in by the programmers to solve the Data-Analysis challenges they encountered, with respect to the KDD process model. When encountering a new but similar KDD problem, this extracted knowledge can be of assistance, by providing suggestions or guidelines.

The main research question is broken down into three sub research questions:

- **RQ 1: What are the requirements of a KDD process model to successfully describe the programming scripts adequately?**

The first research question focuses on identifying a useful model with an appropriate degree of specificity. This model aims to connect the KDD process to the programming scripts. Because the goal is to extract as much knowledge on how programmers solve KDD challenges, the model should describe the scripts as detailed as possible. However, the level of detail is restricted since too much depth would prevent the transfer of the model to other applications and deduct general conclusions.

- **RQ 2: How can parts of scripts be classified with respect to the KDD process model?**

The second research question aims to find a method to classify the scripts with respect to the KDD process model. Every script is viewed as a workflow, with different parts of the script representing different concepts from the KDD process model. Therefore, the assumption is that these parts of code match with the concepts from the model and can be classified as such. With an intended collection of almost 10,000 scripts, this requires the design of an automatic procedure.

- **RQ 3: Having a set of scripts annotated with tasks from the KDD process model, which patterns and insights can be discovered that link to the knowledge that programmers must have put in to solve their challenges?**

The final research question focuses on the analysis of the created dataset. Answering and practising the research questions one and two, results in a set of annotated scripts. These annotated scripts are subject to descriptive analysis. The first analysis is performed to the lengths of blocks of code, representing the parts of scripts. The second, on the co-occurrence of concepts from the model. And the third and last most important analysis studies patterns in the data as discovered by modelling the label of the annotated scripts as state transitions.

1.2 Method
The main goal of this thesis is to describe Data mining programming scripts as instances of the KDD process. These scripts are pieces of code, written by Data Scientist to transform data into knowledge. The approach taken to reach this goal is discussed in this section and visualized in Figure 1-2. For the main approach was started, a collection of scripts is selected which will be used as input for the method followed in this work. This collection should be publicly available such that the
research is repeatable, have enough quality and diversity with respect to the problems solved, and be of a sufficient size. The programming scripts available on Kaggle fulfil all these requirements and were chosen for this purpose. Kaggle is a crowdsourcing platform for predictive modelling and analytics competitions. The open nature of Kaggle ensures diversity in solution since everyone is given the chance to publish his or her solutions on questions pertaining to the available datasets. Sometimes this leads to remarkable results, where solutions proposed by the crowd result in breakthroughs in topics which have already been studied for decades (Rhodes, 2011). On Kaggle, solutions are posted by Data Scientists ranging in experience from novices to well-known experts resulting in approximately 3500 updated solution per day. The value of this community is also stressed by the acquisition of Kaggle by Google on March 2017 (Lardinois, Lynley, & Mannes, 2017).

After preselection on programming language and output, the collection of solutions selected from Kaggle to work on contains 9994 scripts from 6118 different users and over 253 different projects, illustrating the diversity of the collection.

After the selection and collection of the dataset, the next step is to answer the first sub research question; **What is a suitable KDD process model to describe the programming scripts?** In order to answer this question, I performed a literature study on existing KDD models. In the literature, two sorts of models can be identified: process models and ontologies. Both describe the KDD domain, however, process models stay on the abstract levels, where ontologies try to be explicit in their description of the domain. None of the models found in the literature was directly applicable to describe and classify the Kaggle scripts. Therefore, a new adapted KDD process model is constructed which fills the gap between the conceptual model level and the implementation level. As a starting point, I took the concepts and structures from the models as found in the literature survey (Chapman et al., 2000; Fayyad et al., 1996). As visible in Figure 1-1 the conceptual models have Step and Task layers. The new adapted KDD model reuses the step and task layers and is extended with an additional Method layer. During the design of the model, I gave special attention to ensure that the new model was able to classify the scripts. For this reason, I cross-referenced the new model vocabulary of tasks and methods with a representative set of functions, used in data analysis scripts. To reach the final model design, there has been a number of iterations between the design of the model and the classification of the scripts with the new model. During these iterations, not only good prediction scores were taken into account, also the meaningful description of the scripts was evaluated.

With a new suitable model in place, the second research question is ready for answering; **How can parts of scripts be classified with respect to the adapted KDD process model?** The goal is to describe scripts with respect to the process model. Therefore, a method is needed to apply the Steps, Tasks and Methods from the adapted KDD model on parts of the scripts. Because “parts” are relatively vague, “parts” are defined as a single line of code. As the number of scripts and lines of code are too large to process and annotate manually, the process was automated making use of machine learning. The process starts with pre-processing of the data, followed by a transformation step. In this step, the lines of code are transformed in numerical features using word-embedding models. I trained different word-embedding models to find the best transformation model for classification. After the transformation comes the classification step. For classification with machine learning, features and labels are needed. Features are generated during the transformation step. The labels from the adapted KDD model are assigned to a subset code lines using a procedure of manually labelling the function calls in these code lines. Following the idea that the function is the decisive factor for what a line of code is doing. The subset of lines of code used for supervised multi-class classification. Different combinations of transformation models and classification models were tested, to find out which combination was the best to predict the labels. After the classification step,
an evaluation step showed the success of the followed method. A combination of transformation model and classification algorithm was selected, that could predict the label of a line of code with a precision of 81 percent of Step labels to 67 percent on Method labels. Furthermore, the majority of misclassified labels were assigned labels relative similar to the original label or to the unlabelled category, making the predicted labels useful for describing the scripts. This Resulted in a method to classify Python coding scripts with respect to the KDD Process model was used on all lines of code. Resulting in a list of labels for each script, each list represents the workflow followed in the script ready for analysis by the next research question.

Figure 1-2: Overview of method

In the final part of the thesis, a descriptive data-analysis is performed in order to answer the third research question; **Having a set of scripts annotated with tasks from the KDD process model, which**
patterns and insights can be discovered that link to the knowledge that programmers must have put in to solve their challenges? To answer the last research question, three analysis on the set of annotated scripts are performed. Starting with an analysis on the length of consecutive lines of code with identical labels. This analysis showed that these consecutive lines make up a block of code, performing a single task. However, it also shows that there is a need for filtering to make a better analysis of the blocks. The second analysis looks for patterns in the data, by calculating the co-occurrence of the different labels in the scripts. Showing tasks which are more or less likely appear together. The last analysis is that of the entity state transition. The pattern of co-occurrence can be confirmed. Furthermore using the Markov chain model to estimate the probabilities of transferring from one state to any other state, different interesting patterns and insights are discovered.

1.3 Contribution
The main contribution of this work to the field of Knowledge Discovery and Data mining is a better understanding of how Knowledge Extraction problems are solved by data mining workflows. The contributions of the three sub research questions can be summarised as follows:

1. The design of a new KDD process model such that it connects the KDD process and programming scripts. The new model is an aggregation and extension of existing KDD models and has a hierarchical structure of steps, tasks and methods. This adapted model gives a sufficiently detailed description when classifying parts of scripts.
2. A method to classify data analysis scripts in relation to the KDD process was developed. Different machine learning techniques are tested and compared. The method can extend the annotation of code lines with known labels to code lines where the labels are unknown. This multi-label classification problem reaches an accuracy of 67 percent or higher with 34 labels.
3. Analyses of the script gave some interesting insights on the structure of the scripts in relation with the KDD process. First, the general idea that Data Scientist spend the majority of their time on Data cleaning and pre-processing is also reflected in their code. The majority of the code lines in scripts is classified as methods used to process data. Second, it is possible to identify patterns in the data with respect to the process models. Identifying Task and Methods that are often used together. Even more advanced patterns are discovered, by identifying logical next methods after another. This is the first step to help Data Scientist with the discovered knowledge because these patterns can be used as input for recommendations which Methods are usually applied after each other.

1.4 Thesis Outline
This thesis has the following structure. Chapter 2 describes the KDD process model. Starting with existing process models and ontologies. Chapter 3 describes the step taken building a new KDD process model. Ending with a description of the new model. Chapter 4 describes the method developed to annotate the scripts with the entities from the process model following an actual KDD process. In Chapter 4, these workflows are analysed, using different data mining techniques, showing some patterns which extracted from this dataset. The outcomes and methodology are discussed in Chapter 5. Finally, a conclusion is formulated and together with stated in Chapter 6.
2 Theory of Knowledge Discovery and Data mining

When studying data-analysis workflows in scripts, the starting point is a vocabulary to discuss the subject. Even better than a vocabulary, would be a conceptual model that describes the data mining workflows in the scripts. To give it enough support, previous work should provide or back up such a model. This chapter answer the first research question:

- RQ 1: What are the requirements of a KDD process model to successfully describe the programming scripts adequately?

This chapter consists of two sections. The first section describes the literature survey on KDD process models and KDD ontologies. Concluding these models and ontologies are not able to describe the scripts with the required precision. Therefore, the next chapter proposes a new model, incorporating the knowledge from the literature survey and tailored to describe the programming scripts.

2.1 Survey of existing process models and ontologies

2.1.1 Process Models

An adequate model would have a balance between high-level categories that describe the conceptual steps of the KDD process and a connection with the practical application of the scripts. Over the years, several researchers have explored these steps and come up with models to describe them. Kurgan and Musilek (2006) were the first to compare a number of these models, and later work by Mariscal et al. shows that all these models can be seen as offspring from two models, one by Fayyad (Fayyad, Piatetsky-shapiro, & Smyth, 1996) and Crisp-DM(Chapman et al., 2000). Figure 2-1 visualises this evolution of models. The next sections will discuss the outline of these two models, with their purpose, strength and weaknesses.
2.1.1.1 Fayyad’s KDD process model

The KDD process is developed to guide users of Data Mining tools (Fayyad, Piatetsky-shapiro, & Smyth, 1996). Data mining then was an upcoming field, without clear definitions and no model defining how to get from data to knowledge. This model is probably the most important scientific KDD model.

Fayyad describes the KDD process as interactive and iterative involving numerous steps. The model has an outline of nine basic steps:

1. Developing an understanding of the application domain and identifying the goal of the KDD process.
2. Creating a target data set.
3. Data cleaning and pre-processing.
4. Data reduction and projection; finding useful features to represent the data depending on the goal.
5. Matching of the goal to a particular data mining method.
6. Analysing and modelling and hypothesis selection; choosing the data mining algorithms and selecting methods to be used for searching for data-patterns.
7. Data mining, searching for patterns of interest in a particular representation or a set of such representations.
8. Interpreting of mined patterns, this step also includes the visualization of patterns or data.
9. Acting on the discovered knowledge.

From these basic steps, there is a further zoom in on the analysis and data mining step. Depending on the type of goal: verification or discovery, the programmer can choose an applicable method. Whereas verification is limited to hypothesis testing, Fayyad has divided the data mining step in prediction and description. Where predicting tries to find methods to predict the future, description is about finding patterns to represent the data to the user revealing inferred knowledge.

Some of the nine steps are decision or analysis steps and some are focussing on the data transitions. Fayyad provides an outline of the data steps. This is visualised in Figure 2-2. The model stays on a high-level abstraction, without the ambition to cover or describe all the available models and methods. Therefore, a simple comprehensible overview of the basic data transitions is provided.

The KDD process model is simple, making it a natural model to discuss the KDD process. However, its drawback is that it does not make the lower level tasks explicit as it only describes the split up of some steps in abstract tasks. For example, Fayyad does describe the data mining step as divided in prescriptive and descriptive and even giving some examples of possible methods. However, these examples are not formalized, leaving a lot of room for interpretation. This lack of detail makes it difficult to describe programming scripts by Fayyad alone.

Figure 2-2 Outlined process model of Fayad (Fayyad et al., 1996)
2.1.1.2 Crisp-DM

The second big KDD process model is CRISP-DM (Cross-Industry Standard Process for Data mining) which is the most used method for developing data mining projects (kdnuggets, 2004; Mariscal et al., 2010). CRISP-DM has been developed by a group of industrial organizations involved in data mining (Teradata, SPSS, -ISL-, Daimler-Chrysler and OHRA), who saw the need for a broadly applied model.

CRISP-DM is a hierarchical process model, comprising four levels of abstraction: phases, generic task, specialized task and process instances. Each phase consists of generic tasks, which are general enough to cover all possible data mining situations. These generic tasks are further divided into specialized tasks, which describe how tasks should be performed. The process instance level is the recording of actions, for a specific project (Chapman et al., 2000). This hierarchical structure is visible in Figure 2-3.

![Figure 2-3 Four level breakdown of the CRISP-DM methodology](image)

Because the CRISP-DM model stays tool and application neutral, the model describes the phases and generic tasks. Furthermore, the model describes the sequence and the relationship between the phases, for this highest level of phases. The model emphasizes the cyclic nature of the process and the iterative character of the phases, stressing that moving back and forward between the phases is always required. For the generic task, the model does not only provide a description but also the deliverables which make this methodology especially suitable for the management of KDD processes.

The six main phases of the model are:

1. Business understanding, focusing on understanding the objectives of the project.
2. Data understanding, with generic tasks such as data collection, become familiar with the data and discover first insights.
3. Data preparation, which covers all activity to construct the final dataset.
4. Modelling, selecting and applying various modelling techniques. Depending on the technique, sometimes different datasets are needed, requiring an iteration with the data preparation phase.
5. Evaluation: An evaluation of the final model, before going to deployment, to check the alignment with business objectives.
6. Deployment: deploying the model in a live situation or representing the data such that the customer can use it.

A schematic overview of the phases and their transitions is given in Figure 2-4.
One important factor of its success is the fact that CRISP-DM is industry-, tool-, and application-neutral, but there are those who argue against the use of CRISP-DM. First, it tells the user what to do but not how to do it. That is why it does not reach the granularity of the scripts. Second, a major part of the methodology is about reporting and justification, which is completely out of scope when discussing programming scripts.

![Diagram of the CRISP-DM reference model]

Figure 2-4 Phases of the CRISP-DM reference model

2.1.2 KDD Ontologies

Next, in order to process models, I performed a study on KDD ontologies as a formal description of the domain. In contradiction with process models, KDD ontologies attempt to build an understanding of the lower level elements and their relations.

For KDD there are a number of ontologies. While there are general KDD ontologies that cover the entire domain, most of the KDD ontologies aim to optimize planning of data mining workflows. There are ontologies that are even more specific, most of them support particular Data mining workflow software. Three highly cited KDD ontologies are described, which are:

- DMOP, which supports software,
- KDDONTO, designed to optimize workflows,
- OntoDM, which tries to cover the entire domain.

They will be described by their purpose, outline and background, which will give insight into whether these ontologies are adequate to describe scripts.

2.1.2.1 DMOP

DMOP stands for data mining optimization ontology and knowledge base. DMOP is part of the e-lico project which is also known for rapid analytics and rapid miner. DMOP is developed as an ontology for scientific purposes, but its primary goal is to support software. This means that the elements defined in the software limit the described range of entities, which is in the case of rapid miner, quite impressive and extensive.

DMOP is used for rapid miner, but also in Meta learning research, where researchers try to optimize the KDD process. Meta-learning can be described as learning to learn, where they apply machine learning techniques to a collection of meta-data from previous workflows to optimize new
workflows. The other particular feature of DMOP is that instead of treating the data mining algorithms as black boxes, DMOP tries to open up the algorithm, and analyses them in term of their core components.

Based on the KDD process of Fayyad, DMOP covers an entire conceptual framework which covers the following main concepts and their relationships: DM tasks, algorithms, models, datasets, workflows and performance metrics. Figure 2-5 visualizes an overview of the core concepts, where DMKB is the Data mining knowledge database and the DMEX-DB is the Data mining experiment database.

![Figure 2-5 The core concepts of DMOP (Hilario, Nguyen, Do, Woznica, & Kalousis, 2011)](image)

The most important concept of this ontology related to the Kaggle scripts are the DM-tasks, which comprise four major task classes. These tasks are defined by their inputs and outputs. Figure 2-6 visualizes these different tasks. Specializations of these tasks are also defined by their specialized input and output types. This ontology specification goes a lot more into detail than more general ontologies, as they open the algorithms and map all variants found, so they can build the meta-learning on top of it. Because the workflow system rapid miner is linked to it, its implementation results in a very specific definition of the entities, which are described and sorted by their input and output types (Hilario et al., 2011)

Summarizing, DMOP had a good coverage of the domain, from high process level to detailed description of algorithms and a big collection coverage of the existing techniques. However, it is developed as the backbone of a program which is visible in the ontology design. The distinction between entities is made on input and output, which makes the diversity of algorithms high and more detailed than generally defined.
2.1.2.2 KDDONTO

KDDONTO is an ontology designed for supporting both the discovery of suitable KDD algorithms and the composition of KDD processes, which makes it a workflow optimization ontology. It is part of the Knowledge discovery in database virtual mart project (KDDVM).

The designers used an ontology building methodology that is goal oriented. Their building goal is to discover suitable algorithms and the composition of KDD processes. The key concept of KDDONTO is the algorithm. Other fundamental concepts are; method, phase, task, model, dataset, parameter, preconditions/post conditions, performance, and optimization function. The main classes and relationships are visualized in Figure 2-7 (Diamantini, Potena, & Storti, 2009).

KDDONTO defines its phases on the automatable phases from CRISP-DM. This makes it suitable to describe workflows but forces it to break up the CRISP-DM model into automatable and not automatable, which in turn affect the coherence of this model. KDDONTO has implemented 95 classes, 31 relations and more than 140 instances, representing some algorithms of preprocessing, modelling and post processing phases. This gives an idea of the possibilities but is not even an attempt at a full coverage of the possible algorithms.

Summarizing, KDDONTO is a KDD ontology build for workflow optimization. Their focus on algorithms gives a match to the level of detail in the Kaggle scripts. However, the very limited implementation does not reflect all the possibilities in the KDD process domain.
2.1.2.3 OntoDM

The last ontology OntoDM is a general-purpose top-level ontology, whose goal it is to provide a unified framework for data mining. This makes it special, as it is more an ontology for interoperability and discussion than for optimizing a workflow.

OntoDM has a sustained background in broader ontology construction, using top-level ontologies, OBI and EXPO to base its own entities. Related to the higher-level entities, definitions are basic data mining entities such as data type, dataset, data mining task, data mining algorithm, a component of DM algorithm and constraints. It results in what the authors call a deep/heavy weight ontology, with a rigorous meaning for each class, semantically rigorous relations between classes and compliance to a top level ontology (Panov, Dzeroski, & Soldatova, 2008). This is at the same time both the strength and weakness of ontoDM. It has probably the best coverage of the knowledge extraction domain. However, the completeness and its relationship to upper ontologies, with the drive to define everything correctly and consistent, from data entity to programmer, makes the KDD process a minor design element. The broad scope makes it confusing to map concepts from this model to the process.

OntoDM focusses on datasets as the most basic entity, because that is the start of every workflow. Nevertheless, for the relation to Kaggle scripts the most important data mining entity is that of data mining tasks, which is visualized with its properties in Figure 2-8.

Figure 2-7 KDDONTO main classes and relationships (Diamantini et al., 2009)
The above section describes the models found in a literature survey on Knowledge Discovery and Data Analysis. Two sorts of models were studied, KDD process models and KDD ontologies.

The KDD process model has incorporated a data outline, which reflects the transitions of the data from dataset to knowledge, but neglects the level of detail that is needed to reflect the detail in the scripts. Where the Crisp-DM model has its advantages in breaking down the process in smaller process instances. However, its process model also incorporated many theoretical steps that are part of the process, which are not reflected in programming scripts. In addition to this, the specific task and process instances are not defined in the CRISP DM model. This lack of detail makes these high-level process models not adequate for the current purpose. However, they are usefully applied when developing a new model. Whereas the data outline of Fayyad is promising because it fits the data orientation of the scripts, the hierarchical structure of the CRISP-DM gives an idea on how to map higher-level steps on lower-level tasks.

The three ontologies give insight in the field, but they all have their limitations. For DMOP the implementation is too shallow. For KDDONTO, the entity description is too much biased to a specific application. Finally, for OntoDM the ontology wants to describe too much, which makes it very complete, however, the entities are too complicated to match to the functions in the script. Despite this, their implementations are a good starting point to describe the entities encountered in the scripts.

None of the models found in the literature is adequate. The process models are missing the required level of detail, making them to course grained. Moreover, apart from the data outline, which is a part of the model described by Fayyad, they also miss the right focus. The KDD ontologies have a better focus, describing the process with data as centre element. However, all are designed for a specific application, making them too specific. Therefore, they cannot describe the scripts, because they do not have an overlapping vocabulary with the script. Hence, in the next chapter the design of a newly designed model. This model adapts the concepts and structures from the studied models are described, which makes it possible to link the scripts to the conceptual level.
3 Adapted KDD model

This chapter describes the construction of the adapted KDD model. During the literature study, it became clear what was missing in the current conceptual Process models and why the Ontologies did not fit the scripts. This showed the need to adapt the current models and construct an adapted model that is able to describe the scripts.

By describing the construction of the adapted KDD model the first research question is answered:

- **RQ 1: What are the requirements of a KDD process model to successfully describe the programming scripts adequately?**

This chapter first describes the requirements for the adapted KDD model. After that, the different models and information sources that were chosen for construction of the model are discussed. This leads to a basic backbone of the model of different “Steps”. These Steps are split up into “Tasks” and these Tasks are split up into “Methods”. This backbone model needs filling with actual tasks and methods. For this, a set of design rules is used. After describing the design rules, the resulting model is discussed.

### 3.1 Model requirements

From the boundaries and limitations of the current models, the requirements for an adequate model can be derived. Three requirements are formulated to guide the development of an adapted KDD model. These are:

I. The adapted KDD model should be Data-oriented.

Scripts have a focus on both data entities and manipulation of these data entities into new entities, resulting finally in new entities that represent new knowledge. Therefore, the model should have a focus on data, this should be present in the entire model/through all levels of the mode.

II. The granularity of entities described in the model should be one level higher than the level of the entities in the scripts.

Models can describe a process with a different level of detail and granularity. There are very high-level models that use a very coarse description of the process. In addition, there are very detailed models that split up classes to a very high-level of detail. The model needed should have preferably one level of detail higher compared to the level of detail of the subject it is describing. In this thesis, the subject to be described by the model is a collection of data mining scripts, which consists of multiple lines of code, with different functions. The scripts take data and the different functions calls in the lines of code and turn them in a product that represents new knowledge. To be able to classify the different function calls in the scripts the granularity of the model should be one level higher than the granularity of the lines of code and their functions calls.

III. There should be a maximum of overlap between the lowest level of the process model and the content of the scripts.

After defining the focus and the level of detail, the last requirement covers the implementation. The implementation is defined in this context as, what is described and what is not. An adequately aligned model should have a vocabulary that reflects the content being analysed.
3.2 Structure and Sources

This section describes the structure of the adapted model and the sources used to construct it.

The literature study showed that the Fayyad data-outline suits the data orientation and data transformation structure of the scripts, to fulfil the first requirement. This gives the model a high-level framework of steps. The adapted process model starts with this high-level framework of the KDD process. This makes it possible to deduce observations in the scripts, back to the higher-level process.

The Steps need a further breakdown to reflect the same level of detail of the scripts. Therefore, Steps are split up into Tasks and Tasks again into Methods. Tasks are abstract pieces of work, implemented by a data-scientist when writing code. They describe “what” a data-scientist could do. The “Generic Task” level of the CRISP-DM model, described in the previous chapter, is the main inspiration source for this level in the model. The main differences between the adapted model and the earlier studied models are that in the adapted model, the Tasks are all data oriented. With data oriented Steps and Tasks the model fulfils the first requirement, by having the same focus as the scripts.

The Steps and Tasks are the conceptual parts of the adapted model. To reach the required level of granularity to classify the scripts, the Tasks are further split up into Methods. Methods describe “how” something is done, on model or methodology level. In the CRISP-DM model, the Methods are called “Specific Tasks” and are part of the implementation level, however, this level is not specified any further. However, this Method level is needed to connect to the level of granularity of the functions calls in the scripts, grouping functions that use the same generic Method. By reorganizing the Tasks and Methods from three ontologies and reflecting these on a set of functions from data mining programming scripts, the adapted KDD model reflects the data mining domain to a maximum extend. It is however impossible to cover every possible method in the domain. Therefore, every Task holds a Method which is called “Other”. This Other category represents all the Methods that are not used often, borrowed from other domains or even to be newly discovered. The Tasks and Methods which are defined are elicited from the KDD ontology implementations described in the previous section. These are three formal descriptions of the domain, which are all crafted with formal design rules and reasoning. Using them as input gives our model a solid base from previous work. Their implementations all try to model the entire domain or part of it, describing the models and algorithms used in data mining. The choice was made to aggregate the different ontologies manually, following a procedure described below. Combining the different focus areas and generalizations of the three ontologies on Task and Method level. The formalization of the Method level fulfils the second requirement, taking care that the model has the right level of granularity to classify the scripts.
For a successful description of scripts, the model should have a maximum overlap between the lowest level of the adapted KDD model, which is the method level and the used function calls in the scripts. To guarantee this overlap, a large collection of data mining scripts serves as a reference. From these scripts the libraries most often used were selected. Counting the number of imports in the data mining scripts resulted in a list of the most used libraries. Figure 3-2 shows the list of libraries with their number of application/usage and their relative relation to the total number of imported libraries. From this list, the function calls are selected that are actually used in the scripts and are backed up by published API documentation of these function call. The API documentation refers to the code of the libraries and describes the working principle of the function calls. This resulted in a list of more than 900 function calls that are used and described, including a count of the number of applications.

3.3 Model design rules

The adapted KDD model is constructed with a structured approach, following six steps, displayed in Figure 3-3. This structure gives rules to select and aggregate items from the three ontologies, and decide which Tasks and Methods should get a place in the new model. Like any modelling process, this has been an iterative process to achieve the most useful result. The different Steps taken are described and explained below.

**Step 1** retrieves all the relevant data from the selected ontologies. All three ontologies have a published OWL implementation (Diamantini, Potena, & Storti, 2009; “DMOP ontology,” 2009; Pančev
Panov, Soldatova, & Džeroski, 2014). These implementations are visualized using WebVOWL (Lohmann, Link, Marbach, & Negru, 2015), an online ontology visualization tool, to get an understanding of their main entities. WebProtégé was used to dive into the ontologies and retrieve the classes and their implementation on Task and Method level (Musen & Team, 2015). This gave a broad overview of used Tasks and Methods.

**Step 2** is an intermediate step to get a better grip on the unsorted information of Step 1. A first insight in the relevant entities was achieved by sorting the relevant retrieved entities from the ontologies to the steps of Fayyad. This made it easier to start with Step 3.

**Step 3** is a labour-intensive step, which requires a more thorough study of the descriptions of the ontologies. In this step, decisions are made which Tasks and Methods will be part of the adapted KDD process model. For example, if two Methods reflect the same concept, they can be combined. Alternatively, if they mean something different, two different Methods in the same model are needed. Sometimes the decision is made to ignore or skip a Method, either because it is too specific, or because it does not occur in the dataset but is present in the ontologies. As a last alternative, the Method might be better described under another more general Task. These choices are debatable, that is why, after verifying and testing the model, another iteration is made to improve the results of this step.

![Figure 3-3: Model Design Steps](image)

**Step 4** is a manual step and has its own sub procedure to get sustainable labelled functions. During the labelling, the following procedure was followed: First the name of the function was considered, following the rule of thumb that programmers name their function after what it does. Second, if the name gives any reason to doubt its category, the description of the API documentation is evaluated. Third, if the label is still not clear, the open source documentation is evaluated to understand what the function does. This procedure has to be performed by a reasonably skilled person.
Step 5 looks into the distribution and logic of the current labels to the functions in Step 4. Every Method label should at least be applied to five different function calls, and these functions should at least be called in 100 lines of code. Otherwise, the labelled groups get too small which makes it impossible to extend them automatically based on the data. For functions with labels that do not meet these two criteria, a decision had to be made, either to merge them with another label or to put them in an undescribed category called “Other”, which exists for every Task. Thereby, enables the “Other” category the incorporation of Methods that do not exist, because they are not invented yet.

Step 6 reflects the iterative element of the design steps. The output of Step 6 gives insight in the usability of the model for classification. This was used as new input of the model design, starting in Step 3. One example of changes that result from these iterations is the creation of an “Unlabelled” category. This brought a better balance between the occurrences of the steps. Another example is the elimination of Methods with low occurrences, as the machine learning algorithms were not able to learn to recognize them.

The steps described above, give a structured approach that delivers a coherent data oriented model, on the right level of detail. This is not a process that stands on its own but is influenced by the next phase of the project, the classification of the scripts. A couple of iterations were needed to end up with a model, best fitted and suitable to describe the data mining scripts. Figure 3-4 shows the outcome of this process, which is discussed in the next section.
Figure 3-4: The adapted process model
3.4 Developed model

The developed model on Step level

A visualization of the final model is made in Figure 3-4. This section describes the model, discussing its main components and most important decisions made in the design. Special detail is given on the decisions made in the design of the model based on the set of reference functions and the feedback received from testing the model with supervised machine-learning. The description of the model starts on Step level, followed by the Task and Method levels. Table 3-1 illustrates the results of this effort, especially Step 5. The table lists for all elements in the model the unique assigned functions: the number of lines where these functions appear, a percentage of that line in respect to all lines, and all lines with its parent label.

The step level consists of the five categories defined in the data outline of the KDD process model. These categories are; ‘Selecting”, “Pre-processing”, “Transformation”, “Data mining” and “Evaluation”. In the new adapted Process model, these categories are complemented by an “Unlabelled” category for all items that do not belong to one of the other steps. Table 3-1 shows also how many functions exist in each category, how often these functions occur in the scripts, in number and in percentage. The unlabelled category was introduced during the process, to group all functions that do not fit clearly into any of the specified categories. Previously, most of these “Unlabelled” functions were classified as “Pre-Processing”, without a better category available. This resulted in more than 50% of the functions being classified in the “Pre-Processing” category, which creates an unwanted and rather dominant category. Such a dominant category is difficult to handle by classifiers.

Looking into the other categories, the first category is “Selection”, which covers about 10% of the scripts. This happens with only 37 unique functions, which is the lowest number of all categories but is indicating that these functions are often used. In contrast, the “Pre-Processing” category shows the highest number of unique functions, but the number of unique occurrences in the scripts is much lower. The “Data mining” category shows again a different outcome to these previous two categories, it shows just a small percentage of the script lines but many unique functions. This indicates that data mining functions are tailored for the task.

Proceeding to the task level in each category, it immediately stands out that under many Tasks there is a method called “Other”. The “Other” methods contain all functions that cannot be assigned to previously defined methods and there are not enough of these functions to justify a separate Method.

The developed model on task and method level

The model starts with the first Step in the process. “Selection” only consists of one Input / Output (IO) category, which looks minimal. On the other hand, the three ontologies do not even have a selection category because this is out of their scope. However, the study of the scripts showed that each script always begins with reading a data set; therefore, “Selection” is defined in this model to classify these script lines.

The second Step “Pre-Processing” is divided into two Tasks, “Data-Pre-processing” and “Data-Sampling”. Data-Pre-processing contains all the Methods that are used to prepare data before modelling. For “Data-Pre-processing” the specific Methods “Data Cleaning” and the “Removal of missing values” are defined. “Data-Pre-processing” also contains a method for “Other” which is the largest of unique functions in the model. In this “Other” methods category are all the functions gathered that are clearly used for processing data, however, these functions are too general to
classify the method specifically. “Data sampling”, on the other hand, is a straightforward task: it holds all Methods that divide the data set into different subsets, such that training and testing can take place on independent data sets. In an earlier version of the model, there was also a “Data-Sampling” section under the evaluation Step, because it is partly how the evaluation of a data mining script is organized. For consistency and because the functions look very similar, all sampling is grouped together under “Pre-processing”.

The third Step “Transformation” is divided into three Tasks: “Transformation”, “Selection” and “Extraction”. “Transformation” is performing a mathematical action on any observation of a variable or data set. “Selection” is selecting existing features that are included in the rest of the process. “Extraction” is calculating new features from existing features. The classification of these Tasks and Methods was ambiguous. In other models and ontologies, the category “Transformation” is often combined with “Pre-processing”, as the distinction between the Steps is not always clear. However, “Transformation” and its tasks are kept as part of the model because the process of selecting useful features is a decisive Step in the KDD process. The initial model of Fayyad contains the two steps separately, therefore it is part of the adopted higher-level model. Furthermore, only functions that are clearly part of the “Transformation” step are classified as such, other functions are classified as Pre-processing.

The fourth Step is “Machine learning”. Splitting up “Machine-Learning” is not intuitive, because there is not a list of Tasks that describe what to do to complete this Step. Therefore the choice was made to divide the Step not in Tasks, but into general container Methods, which group algorithms often used in Machine Learning. These container Methods are split up again in more specific Methods, as usual in the other Steps. Other choices could have been made too. For example, another common way to split up Machine Learning is to divide the algorithms between prescriptive and descriptive machine learning, which did not come naturally with many functions as often both are used simultaneously. For example, regression functions are used to describe trends in data, but also to predict new values. Therefore it was decided to ignore the prescriptive/descriptive layout and divide the tasks in the following methods: “Classification”, “Clustering”, “Ensemble” and “Regression”.

The final Step, Interpretation, is divided into “Evaluation”, “Visualization” and “IO”. This is the Step where machine-learning models are transformed into new knowledge. This can be done by “Evaluation”, such as a calculation of the accuracy of the calculated categorization. Another possibility is visualizing the results to give new insights. The last option is outputting the calculated values, which often happens to make a contribution for a competition. All of these three Tasks are reflected in the model and are not further specified as the amount of unique function was considered to be too limited.

3.5 Conclusion
Within a literature survey, no adequate models were found suitable to describe the script. A study of the existing model has led to the requirements needed for the development of an adapted model. A new adapted KDD model was built, with the use of process models, ontologies and data from data mining scripts. This results in an improved process model. This new process model has a hierarchical structure, with entities on Step, Task and Method level. Its design follows the data outline of the Fayyad model, but it is adapted, such that it fits the functions used in Data-dining scripts. This model will be deployed in the next Chapter.
<table>
<thead>
<tr>
<th>Category</th>
<th>Unique functions</th>
<th>Function appearances in scripts</th>
<th>Percentage appearance above layer</th>
<th>Percentage appearances total</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECTION</td>
<td>37</td>
<td>16677</td>
<td></td>
<td>11.74%</td>
</tr>
<tr>
<td>IO</td>
<td>37</td>
<td>16677</td>
<td>100.00%</td>
<td>11.74%</td>
</tr>
<tr>
<td>PRE-PROCESSING</td>
<td>254</td>
<td>42486</td>
<td></td>
<td>29.91%</td>
</tr>
<tr>
<td>Data-Processing</td>
<td>217</td>
<td>39653</td>
<td>93.33%</td>
<td>27.91%</td>
</tr>
<tr>
<td>Cleaning</td>
<td>6</td>
<td>10361</td>
<td>26.13%</td>
<td>7.29%</td>
</tr>
<tr>
<td>Remove missing values</td>
<td>8</td>
<td>1716</td>
<td>4.33%</td>
<td>1.21%</td>
</tr>
<tr>
<td>Other</td>
<td>203</td>
<td>27576</td>
<td>69.54%</td>
<td>19.41%</td>
</tr>
<tr>
<td>Dataset sampling</td>
<td>37</td>
<td>2833</td>
<td>6.67%</td>
<td>1.99%</td>
</tr>
<tr>
<td>Cross validation</td>
<td>9</td>
<td>138</td>
<td>4.87%</td>
<td>0.14%</td>
</tr>
<tr>
<td>Random</td>
<td>12</td>
<td>1467</td>
<td>51.78%</td>
<td>1.03%</td>
</tr>
<tr>
<td>Train test split</td>
<td>16</td>
<td>1228</td>
<td>43.35%</td>
<td>0.86%</td>
</tr>
<tr>
<td>TRANSFORMATION</td>
<td>105</td>
<td>8545</td>
<td></td>
<td>6.02%</td>
</tr>
<tr>
<td>Extraction</td>
<td>48</td>
<td>2387</td>
<td>27.93%</td>
<td>1.68%</td>
</tr>
<tr>
<td>PCA</td>
<td>14</td>
<td>720</td>
<td>30.16%</td>
<td>0.51%</td>
</tr>
<tr>
<td>SVD</td>
<td>5</td>
<td>230</td>
<td>9.64%</td>
<td>0.16%</td>
</tr>
<tr>
<td>Other</td>
<td>29</td>
<td>1437</td>
<td>60.20%</td>
<td>1.01%</td>
</tr>
<tr>
<td>Selection</td>
<td>26</td>
<td>733</td>
<td>8.58%</td>
<td>0.52%</td>
</tr>
<tr>
<td>Hypothesis test based</td>
<td>2</td>
<td>194</td>
<td>26.47%</td>
<td>0.14%</td>
</tr>
<tr>
<td>Univariate</td>
<td>18</td>
<td>428</td>
<td>58.39%</td>
<td>0.30%</td>
</tr>
<tr>
<td>Other</td>
<td>6</td>
<td>111</td>
<td>15.14%</td>
<td>0.08%</td>
</tr>
<tr>
<td>Transformation</td>
<td>31</td>
<td>5425</td>
<td>63.49%</td>
<td>3.82%</td>
</tr>
<tr>
<td>Discretization</td>
<td>8</td>
<td>664</td>
<td>12.24%</td>
<td>0.47%</td>
</tr>
<tr>
<td>Normalization</td>
<td>8</td>
<td>3088</td>
<td>56.92%</td>
<td>2.17%</td>
</tr>
<tr>
<td>Scale</td>
<td>10</td>
<td>1611</td>
<td>29.70%</td>
<td>1.13%</td>
</tr>
<tr>
<td>Standardization</td>
<td>5</td>
<td>62</td>
<td>1.14%</td>
<td>0.04%</td>
</tr>
<tr>
<td>DATA MINING</td>
<td>227</td>
<td>12533</td>
<td></td>
<td>8.82%</td>
</tr>
<tr>
<td>Classification</td>
<td>101</td>
<td>5852</td>
<td>46.69%</td>
<td>4.12%</td>
</tr>
<tr>
<td>Bayes</td>
<td>15</td>
<td>339</td>
<td>5.79%</td>
<td>0.24%</td>
</tr>
<tr>
<td>KNN</td>
<td>18</td>
<td>570</td>
<td>9.74%</td>
<td>0.40%</td>
</tr>
<tr>
<td>LDA</td>
<td>7</td>
<td>103</td>
<td>1.76%</td>
<td>0.07%</td>
</tr>
<tr>
<td>Neural Network</td>
<td>8</td>
<td>165</td>
<td>2.82%</td>
<td>0.12%</td>
</tr>
<tr>
<td>Random Forest</td>
<td>8</td>
<td>2262</td>
<td>38.65%</td>
<td>1.59%</td>
</tr>
<tr>
<td>SVC/SVM</td>
<td>20</td>
<td>685</td>
<td>11.71%</td>
<td>0.48%</td>
</tr>
<tr>
<td>Tree</td>
<td>12</td>
<td>608</td>
<td>10.39%</td>
<td>0.43%</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>13</td>
<td>1120</td>
<td>19.14%</td>
<td>0.79%</td>
</tr>
<tr>
<td>Clustering</td>
<td>14</td>
<td>205</td>
<td>1.64%</td>
<td>0.14%</td>
</tr>
<tr>
<td>Ensemble</td>
<td>44</td>
<td>5310</td>
<td>42.37%</td>
<td>3.74%</td>
</tr>
<tr>
<td>Boosting</td>
<td>36</td>
<td>5236</td>
<td>98.61%</td>
<td>3.69%</td>
</tr>
<tr>
<td>Combining</td>
<td>8</td>
<td>74</td>
<td>1.39%</td>
<td>0.05%</td>
</tr>
<tr>
<td>Regression</td>
<td>68</td>
<td>1166</td>
<td>9.30%</td>
<td>0.82%</td>
</tr>
<tr>
<td>Linear</td>
<td>34</td>
<td>464</td>
<td>39.79%</td>
<td>0.33%</td>
</tr>
<tr>
<td>Non-linear</td>
<td>34</td>
<td>702</td>
<td>60.21%</td>
<td>0.49%</td>
</tr>
<tr>
<td>INTERPRETATION</td>
<td>157</td>
<td>8719</td>
<td></td>
<td>6.14%</td>
</tr>
<tr>
<td>Evaluation</td>
<td>64</td>
<td>2721</td>
<td>31.21%</td>
<td>1.92%</td>
</tr>
<tr>
<td>IO</td>
<td>20</td>
<td>4486</td>
<td>51.45%</td>
<td>3.16%</td>
</tr>
<tr>
<td>Visualization</td>
<td>73</td>
<td>1512</td>
<td>17.34%</td>
<td>1.06%</td>
</tr>
<tr>
<td>UNLABELLED</td>
<td>166</td>
<td>53100</td>
<td></td>
<td>37.38%</td>
</tr>
</tbody>
</table>
4 Classification of scripts

The previous chapter ended with an adapted process model to describe scripts. In this chapter, this adapted KDD process model is applied on Data mining programming scripts. More precisely the goal is to classify scripts lines because the annotated script lines make it possible to analyse how Data Scientist apply the Tasks and Methods from the model. This chapter describes the workflow used to achieve the classification, providing an answer to the second research question:

- RC2: How can script lines be classified with respect to the Adapted KDD process model?

The goal is to develop a method that can classify parts of unlabelled scripts with respect to the adapted process model. This will result in a list of labels that represents the KDD workflow used in that script. The work described follows the steps of the KDD process. This process is evaluated by training a classification model on labelled data and tested by measuring the extendibility of the model to untrained data. After this evaluation, the model will be applied to the entire dataset. The model labels all code lines with a label from the adapted process model, resulting in a collection of annotated scripts.

4.1 Selection of Target Data

The dataset used is a collection of the Python scripts scraped from Kaggle.com. Kaggle has programming scripts, which they call kernels, written in four programming languages: Python, R, SQLite and Julia. On the 11th of January 2017, all available kernels were scraped from Kaggle. Table 4-1 shows these scraped kernels by their programming language and indicates that the majority are Python and R kernels. The different kernels are too diverse to process them all at the same time. Therefore only Python kernels were processed at first. Python and R are both popular languages within data science. However, Python has a few advantages when processing the scripts. First, it is a language designed to be easy to read and understand, which makes it easier to process. Second, in contrast with R, in Python, it is only possible to execute a function in one way, which makes the syntax easier to understand and process. A third and final argument was that the set of Python kernels is the largest set. Python and R kernels are available in two formats: scripts and notebooks. From the 19029 Python kernels, 9033 were notebooks and 9996 were in script format. A second selection is made by choosing scripts over notebooks. First, because scripts are easier to process, scripts do not have an extra shell formed by an XML layer taking care of the formatting as notebooks have. Second, the scripts focus on code, with a few lines of comment, while notebooks are formed as demonstrations with more natural language in between the code. This thesis focuses on studying code and leaves comments out of scope. This because, comments are very personal depending on the purpose, coding style and maturity of the script.

Table 4-1: Scraped scripts split up by programming language

<table>
<thead>
<tr>
<th>Language</th>
<th>Kernels</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>19029</td>
<td>62.8 %</td>
</tr>
<tr>
<td>R</td>
<td>10614</td>
<td>35.0 %</td>
</tr>
<tr>
<td>SQLite</td>
<td>544</td>
<td>1.8 %</td>
</tr>
<tr>
<td>Julia</td>
<td>110</td>
<td>0.4 %</td>
</tr>
<tr>
<td>All</td>
<td>30297</td>
<td>100.0 %</td>
</tr>
</tbody>
</table>
4.2 Pre-processing

The next step, after having an adapted process model and a selected set of scripts, is to prepare these scripts to apply the model on it. The goal of this process is to find blocks of code that represent the conceptual step, tasks and methods from this process model. There is no rule to define a block of code. In the best case code is structured clearly, enabling humans to understand how the code is working, by looking at the format. In clear written code, every part is properly commented and pieces of code are separated by blank lines, separating code working on other tasks. Unfortunately, most of the time this clear formatting is not in place. And every programmer follows his own rules to decide which lines are blank. The only consistent block of code that can be found is a line. Therefore, the problem is simplified. This section tries to answer if it is possible to classify a single code line with respect to the process model.

With single lines of code as the initial target for labelling, a method has to be developed to attach process models labels to these lines. The heuristic to solve this problem is as follows. Given a line of Python code, only function calls are relevant to classify it. Because the function call is the element in the line of code that defines the relevant transformation from the data. Furthermore, each (relevant) function call can be clearly associated with a Step, Task and Method from the process model. Table 4-2 illustrates this labelling of code, where for some lines of code, the function call is identified. After identification, the function call gets a label. This label is attached to the line of code. This heuristic is furthermore back-up by the set of labelled function calls, which come from the previous Chapter. There a collection of relevant function calls is manually labelled with respect to the adapted process model. From a set of more than 900 labelled function calls, 82 percent of the functions are categorized to at least the conceptual Step in the adapted process model. The remaining 18% gets an “Unlabelled” tag. Together these labelled functions cover 52 percent of the lines with identified functions in the collection of Python scripts.

Table 4-2: Examples of labelled lines of Code

<table>
<thead>
<tr>
<th>Line of Code 1</th>
<th>train_csv = pd.read_csv(train_path)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant library and function call</td>
<td>Pandas and <code>read_csv</code></td>
</tr>
<tr>
<td>Step: Selection</td>
<td>Task: <code>IO</code></td>
</tr>
<tr>
<td>Line of Code 2</td>
<td>x_train, x_valid, y_train, y_valid = train_test_split (x_train, y_train, test_size=0.1, random_state=4242)</td>
</tr>
<tr>
<td>Relevant library and function call</td>
<td>Sklearn and <code>train_test_split</code></td>
</tr>
<tr>
<td>Step: Pre-processing</td>
<td>Task: <code>Data-Sampling</code></td>
</tr>
<tr>
<td>Line of Code 3</td>
<td>clf = xgb.train(params, d_train, 1000, watchlist, early_stopping_rounds=20)</td>
</tr>
<tr>
<td>Relevant library and function call</td>
<td>Xgboost and <code>train</code></td>
</tr>
<tr>
<td>Step: Data mining</td>
<td>Task: <code>Ensemble</code></td>
</tr>
</tbody>
</table>

However, to reach a full understanding of the workflow in the script, this set of labelled function calls is not enough. It would be better if it were possible to extend the labels from the lines with known function calls to lines with unknown functions calls. Such that, if unknown functions calls are encountered in a script, it is possible to incorporate these in the analysis. It is not possible to manually annotate all the function calls in the collection of scripts. Therefore an automatic method to execute classification is needed. This automatic classification is performed by supervised classification and will be described in Section 4.4. However, for this classification, a feature set is needed. This feature set will be built by extracting features from the lines of code. This transformation of the data by feature extraction will be described in the next section.

4.3 Transformation by feature extraction

The next step in the process is transformation, extracting useful features from the pre-processed data. For machine learning, features are needed to describe the items, as input for the classification
models. The target objects are lines of code; therefore, features are needed for every line of code. Word-embedding techniques, developed within the field of Natural Language Processing, have been chosen to generate these features. The heuristic for this is that lines of code that execute a same sort of operation are expected to have similar semantics. This would make it possible, that when encountering an unlabelled line of code, which is semantically similar to a labelled line of code, the label can be extended to this unlabelled line of code.

The first step is building a feature set that has learned the semantics of code lines. Within the feature set, the expectation is that given a line of code with a certain feature vector, a semantic similar line of code should have a similar feature vector. To calculate these feature vectors, the state of the art Word2Vec algorithm is used in different variants. Word2Vec is proven to be a successful algorithm to model semantic similarity (Goldberg et al., 2014). Different models are built and evaluated to discover the best scoring model. In this section, first the theory about word embedding and the used models: Word2Vec, Doc2Vec and Python2Vec will be discussed, followed by a description of the build word embedding models and ending with an evaluation of these Word2Vec models.

4.3.1 Word-embedding

Word Embedding is a collection of Natural Language Processing Techniques (NLP) where words are mapped to a vector. The general idea behind word embedding is that a word is characterized by the words surrounding it. Advantages of word-embedding compared to other NLP techniques is that word-embedding takes into account the semantics, and from a computation perspective it works with relatively small and dense vectors.

A renowned word-embedding example, which is a good illustration of the power of a working model, is that of “Man” stands to “King” as “Woman” stands to “???”. Which is an appealing example where with a trained model the calculation holds that: vector (“Man”) – vector (“King”) + vector (“Women”) results in vector (“Queen”). (Mikolov, Yih, & Zweig, 2013). Next, to this example, other research showed that the Word-embedding models are also useful in other applications, such as topic modelling and ranking (Bhattacharjee & Petzold, 2017; Nakov, 2016)

The promising results within the field of NLP by word embedding are mainly achieved with the Word2Vec implementation. Further research on this implementation, shows that is also possible to process paragraphs and documents in a similar way, which is called Paragraph2Vec or Doc2Vec (Le & Mikolov, 2014). Finally, there is already some research on word-embedding on the code, showing that word-embedding can be applied successfully because the code has semantics too, this is called Python2Vec (Gude, 2017).

4.3.2 Word2Vec

Word2Vec is a specific implementation or algorithm of word embedding, developed by Google, with Mikolov as the main contributor (Mikolov et al., 2013). Word2Vec learns the vectors with a machine learning technique, using a recurrent shallow neural network. The training set of documents is processed to a set of unique words, called the vocabulary. Every unique word in the training set is assigned a vector. The algorithms use a sliding window, which goes over the sentences, trying to predict the next word with the vectors from the other words in the window. There are two variants of the algorithm: CBOW and Skip-Gram. Continues bag of words (CBOW) is used to predict the vector of the target word, with the vectors of the surrounding words. CBOW is not taking into account the distance to the target word. Skip-Gram is predicting the surrounding words, with the vector of the target word, taking into account the distance to the target word. The experiment shows that CBOW is faster, but that Skip-gram gives better results for infrequent words.
Word2Vec has its main application in recommender systems and topic modelling. Normal configurations use sizes of 5 to 10 words for the sliding window, making passes over the dataset at least 5 to 10 times, as they calculate feature vectors with a length between 100 and 300.

Word2Vec shows good results in NLP competitions. (Liu, Liu, Chua, & Sun, 2015; Nakov, 2016), and there are well developed and supported libraries in place, such as Gensim, Tensorflow and Deeplearning4J (Deeplearning4j Development Team., 2017; GoogleResearch, 2015; Mikolov & Com, 2014). The disadvantage of Word2Vec is, that most of the time, it is implemented as a black box. Word2Vec gives good results, however, the mathematical foundation is still weak to explain how it operates. Nonetheless, in recent years, progress is being made to get it better understood (Goldberg et al., 2014).

### 4.3.3 Doc2Vec

Doc2Vec builds on Word2Vec and is a follow up of the word-embedding application. Doc2Vec is sometimes called Paragraph2vec, depending on the input. In this application, the input of the algorithm are not single words, but paragraphs or documents. The fundament of Doc2Vec is still a Word2Vec model, but now the documents are assigned vectors as well. Which makes it possible to measure the similarity between documents. The feature vector of this document can now be used as input to cluster these documents, find related articles or use them in recommender systems.

Again, the application comes with two implementation algorithms: PV-DM and PV-DBOW in Paragraph vector-Distributed Memory (PV-DM), the paragraph token and its vector act as an extra word, which is taken into account when estimating a word vector. Therefore, it holds the information of what is missing in the topic. The big advantage of PV-DM is that it is trained on unlabelled data. Therefore, it still works if there is no labelled data. In Distributed Bag of Words (DBOW), the context of the paragraph is not taken into account. Here the document vector is used to predict the missing word, taking a random selection of the words in the paragraph, not taking into account the word order. Where PV-DM gives a state of the art performance is some cases, DBOW outperformances it, despite being a simpler model (Le & Mikolov, 2014).

The advantage of Doc2Vec to other techniques is its ability to process segments of text with variable length, where other methods are limited to a fixed number of words. The disadvantage of Doc2Vec, that it has in common with other NLP techniques that process documents, it that it is sensitive to large variations in length and that it is also sensitive to the sort of text. For example, spoken text transcriptions and manuals use different semantics and vocabulary, which degrades performance when extending one trained model on a different data set.

### 4.3.4 Py2Vec

This study is not about natural language, but about programming scripts. Programming scripts have syntax and semantics too, so the aim is to extend Word2Vec to programming scripts. There is already work done by “Lab 41”, which is a research group that works to support the U.S. Intelligence Community. They applied Word2Vec on Python, calling it Python2Vec (Gude, 2017). Gude trained Python2Vec on the code of more than 20 popular Python libraries. He found that this implementation is able to find sound related entities. The tests show that calculating the 10 most similar objects to a function results in similar functions. However, he did not explore the possibilities to implement this in a real application.

### 4.3.5 Building of the Word2Vec models

This paragraph describes the design of the Word2Vec model built. The resulting model makes it possible to calculate a feature vector for every word in the collection of Kaggle scripts. These feature
vectors are needed as input for machine learning classification. The Word2Vec algorithm calculated vectors for every word in the dataset, whereby words with similar semantics get a similar vector, the model that perform best in this task will also be the model giving the best result when running the classification algorithms. Because it is unknown which approach delivers the model that represents the semantic similarity the most accurate, two different models are built, both with their own heuristic.

The first model is named Python2Vec, which follows the main idea that code has semantics. These semantics when quantified hold the information to calculate a performing similarity model. Using the Gensim library, a model is built with the lines of code in the Python scripts as input. The following procedure has been followed. During the preprocessing, all the blank lines and lines with comments were filtered out. For every line of code, all the words are elicited and put in an array, while filtering out all the symbols. An example how a line is transformed into an array is given in Figure 4-1. These arrays were the input of the Word2Vec algorithm. This Word2Vec algorithm was initialized with the following tuning parameter, 5 passes over the data, output vector size of 100, min count of 10, sliding window size of 5, using CBOW algorithm and an alpha or learning rate of 0.025.

```python
# Comment line which is ignored
sss = StratifiedShuffleSplit(n_splits = 2, random_state = 42, test_size = 0.25)

# Result in an array for every line like this
[sss, StratifiedShuffleSplit, n_splits, 2, random_state, 42, test_size, 0.25]
```

*Figure 4-1: An example of the filtering of the line of code*

The second model is called StackOverFlow2Vec. The main idea behind this model is that a collection of not even 10.000 documents is a small basis for a Word2Vec model. Word2Vec models work better on very large collections. Therefore, a second model was developed with a much larger collection of sentences. This larger collection of sentences was found on StackOverFlow\(^2\). StackOverFlow is a question and answer website specialized in questions about programming languages. By selecting all the Python related question the vocabulary stayed close to the lines of code from the Data mining programming scripts that were targeted. This scraped collection of question and answer post gives a bigger foundation for the vector model. All Python questions and answers were downloaded, resulting in a collection of more than a million records. From these records, a Word2vec model was built using the same parameters of the Python2Vec model.

### 4.4 Data mining

The next step in the process is data mining. In data mining, the extraction features of the transformation step are applied to the data. With two models in place, which hold features for the words, the next step is to evaluate the quality of these models on their ability to represent the semantic similarity. The project started with in unsupervised classification approach, which made it really difficult to judge the resulting model. Therefore the project continued with supervised classification approach, which gave better results in evaluating and comparing the models. Both approaches methods are discussed below.

#### 4.4.1 Unsupervised Classification

With two models build, it was the aim to compare their semantic similarity implementation. However, without an independent reference set, it was difficult to give sustained judgements on the

\(^2\)stackoverflow.com
quality of embedding the semantic similarity. The common way to make such a judgement within NLP, if there is no reference set available, is to take entities from the vocabulary and calculate the ten most similar entities in the vocabulary. A common distance measure for this is the cosine distance. This following procedure was adopted, for the 50 most used functions; the ten most similar entities in the different vocabularies were calculated. Now, these results had to be judged manually on their similarity. Which was the point the experiment came to a hold because it was not possible to find an unambiguous and comprehensive definition of semantic similarity. Without this definition, it appeared not possible to judge the generated similarity calculations.

Still wanting to try, an attempt was taken to give a binominal judgment if a similarity calculation was correct or not. The following rule to judge semantic similarity between input and output was created: “Two entities in a script are semantically similar when they contain two lines that have the same goal. They are expected to be in the same role in those script lines.” While being aware that this definition is ambiguous, the expectation is that a more sustained judgement on the models could be collected. Examples of clear semantic similar entities are “Integer” and “Double”, both defining the type of a variable. Another example is “Shape” and “Length”, both requesting a dimension property of a variable. Clearly not semantically similar are: “Dataframe” and “AgeIsNull”, where the first is a data structure and the other a description. However, most of the results are not this clear. Two words can be similar in many ways. Moreover, the similarity results that were calculated, showed a lot of them. When investigating specific results, most of the times a logic relation was found, which explained the high similarity in the model. However, these relations were very diverse and it was not possible to put them in one definition. All effort resulted in the idea that the trained models do hold semantic similarity, but there is no explicit way to judge which model reflects semantic similarity best. Therefore both models are taken to the step of supervised classification and its evaluation.

4.4.2 Supervised Classification
Supervised classification gives a way to evaluate the Word2Vec models and to verify if the labels can be extended to unknown function calls. There is a set of labelled function calls; this set is used to apply labels on the code lines. These experiments aim to evaluate if and how it is possible to extend these labels to lines of code with unknown and unlabelled functions. The two Word2Vec models available are expected to embed semantic similarity. The experiment set up has two machine learning algorithms, Neural Network and Random Forest that are known to perform well on complex datasets, such as the available Word2Vec models. The algorithms will be trained on the multi-nominal labels. Furthermore, they are trained on the different label levels, step, task and method. For evaluation, precision, recall and F1 score are calculated to evaluate this multi-label classification problem. In addition, a confusion matrix will show how the misclassified labels are spread over the other labels. This makes it possible to evaluate if the labels can be extended and to select the best performing combination of model and algorithm.

The input of the classifiers is an aggregation of the available models and data. For every line of code, with a function in the labelled set, the feature vector is calculated. Taking the average of the word feature vectors in that line. The resulting set is split up to perform 10-Fold cross validation, taking into account that feature vectors linked to similar functions should be in the same set. This prevents testing on data that is first used for training. Using 10-fold cross validation, the machine learning algorithms are trained ten times, leaving out one tenth of the data for testing. The results of the test are averaged. For training, the classification algorithms of SkiKitLearn (Pedregosa F. et al., 2011) are used. The following algorithm setting was used. The Neural Network algorithm is initialized with the ‘adam’ solver and 100 iterations. The Random Forest is initiated with 29 estimations. The other
parameters were set by default. The ‘adam’ solver is known to give good results on complex datasets as ours. (Pedregosa Fabian et al., 2011). After some preliminary testing 100 iterations and 29 estimations were settings where the results reached their average maximum.

4.4.3 Evaluation

The last step of the KDD process is an evaluation of the supervised classification. This evaluation shows if and which chosen models and algorithms can classify script lines adequately. First, the best combination of feature model and machine learning algorithm is selected, by comparing the precision recall and F1 score. On the selected combination, a further analysis is done of the misclassification, by calculating and describing the confusion matrix.

For the trained classification models, the calculated precision, recall and F1 score are visible in Table 4-3. The results show that the Neural Network performs best. The Random Forest only scores best on the step and task in combination with the Python2Vec model. Furthermore, Python2Vec model outperforms the StackOverflow model. Zooming in on Python2Vec and Neural Network, as best performing combination, I can say that the classification outcome is reasonably good. Especially if you take into account that it is multinomial classification, with 6, 16 and 35 labels, where a random classifier has an expected result of 0.16, 0.06 and 0.03.

<table>
<thead>
<tr>
<th>Table 4-3: Average result of the classification with Word2Vec models</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Python2Vec</strong></td>
</tr>
<tr>
<td><strong>Neural Network</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Random Forest</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The resulting best combination of algorithm and feature model is a neural network with Python2Vec. For this model on method level, the confusion matrix is calculated and which can be seen in Table 4-6. The rows indicate the true label and the columns the assigned label. Furthermore, the methods are grouped by Step and Task. Some elements of this confusion matrix are remarkable. First, the diagonal of the matrix is clearly highlighted, which represents the correctly classified labels. Second, misclassification on method level is still in the same step, so this is much better than classifying towards another step, making the classification quality reasonably good. This indicates that if a method is misclassified, the assigned method is wrong, but the assigned step is probably still correct. This gives the misclassification of code lines a lower impact than a wrongly assigned step. Third, the columns “Unlabelled” and “Pre-processing_data-processing_other” have higher numbers than the other columns. This means that if a line of code is misclassified, it will likely be labelled with one of those two method categories. This is also an improvement on a fully misclassified line because these are leftover categories, which are likely to be left out later on when analysing workflows. All these results together show that the developed method can indeed be used to classify scripts.
4.5 Classification of all the script lines

The results from the previous sections show that the list of labelled functions could be extended reasonably to other labels. Based on these results, a classification model was trained with on lines of code which contains a labelled function call. The classification model has been trained using the Python2Vec model and Neural Network. This model is applied to all the lines of code in the scripts.

Providing a classification on the three levels (step, task, and method), for all lines. On this collection of classified lines of code, some numerical analysis is performed, followed by a more extended descriptive analysis in the next chapter.

4.5.1 Numerical Analysis

By transferring the model to all lines of code, also the unlabelled lines of code, the ground truth available in the last chapter is lost. This makes it impossible to validate the model completely as in Table 4-3. However, it is possible to evaluate the model to some extent. First, the performance of the model on the training data will be analysed. Secondly, the distribution of the different classes will be displayed. Third and last, the distribution over the classes between the labelled and unlabelled data will be analysed.

Displayed in Table 4-4 are the precision, recall and F1 score calculated on the code lines that were used as input for the model. With 54% of the lines of code provided with a ground truth label, the labels for at least half the lines of code are known to be correct. For the remaining 46% of the lines of code, the expectation is that the precision will come near the numbers in Table 4-3. These numbers make the expected precision on method level grow towards 0.83.

*Table 4-4: Results of classification of full dataset.*

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1 score</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEP</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>TASK</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>METHOD</td>
<td>0.97</td>
<td>0.97</td>
<td>0.97</td>
</tr>
</tbody>
</table>

It is also possible to learn something from the misclassification of the labelled lines. Table 4-5, shows the confusion matrix of the classified label and the true label on step level. As expected, the highest numbers are on the diagonal, representing the high precision score. However, if there is misclassification, there is probably a misclassification to “Unlabelled” or to “Pre-processing”. This makes a misclassification less sincere because these classes can be filtered out later in the analysis.

Table 4-5 on step level is representative of the other results on task and method level. The assumption is that when classifying lines of code that should be “Unlabelled”, the misclassification will follow the same pattern. Other results will strengthen this assumption in the further analysis.
Table 4-5: Confusion matrix of the classified labels on Step level

<table>
<thead>
<tr>
<th>TRUE LABEL</th>
<th>Selection</th>
<th>Pre-processing</th>
<th>Transformation</th>
<th>Data mining</th>
<th>Interpretation</th>
<th>Unlabelled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>15156</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>Pre-processing</td>
<td>17</td>
<td>35811</td>
<td>16</td>
<td>58</td>
<td>15</td>
<td>355</td>
</tr>
<tr>
<td>Transformation</td>
<td>2</td>
<td>30</td>
<td>5871</td>
<td>15</td>
<td>3</td>
<td>132</td>
</tr>
<tr>
<td>Data mining</td>
<td>0</td>
<td>28</td>
<td>8</td>
<td>8892</td>
<td>18</td>
<td>27</td>
</tr>
<tr>
<td>Interpretation</td>
<td>3</td>
<td>20</td>
<td>5</td>
<td>73</td>
<td>4917</td>
<td>27</td>
</tr>
<tr>
<td>Unlabelled</td>
<td>12</td>
<td>660</td>
<td>126</td>
<td>21</td>
<td>23</td>
<td>33192</td>
</tr>
</tbody>
</table>

Now the focus is extended to the entire labelled data set. Figure 4-3 visualizes the distribution of the labels over the different classes. Things to notice it that “Unlabelled” takes a major part of the assigned labels. Making it not possible to say something about these. The second largest class is pre-processing. Not strange, because it is known that data-scientists spend most of their time on preparing data. The big chunk of labels assigned to the step “Other” and “Pre-processing”, give the idea that this will give some clutter later in the analysis. Finally, it is remarkable that some method labels are assigned to just a small number of lines of code. Resulting in small angles in the diagram and accordingly small percentages of the labelled lines. However, this is a result of the model, which folds many of such classes.

![Figure 4-2: Distribution of the labels over the Step classes. The blue bars represent the lines of code with labels and the red bars the lines of code without labels.](image)

The last analysis to be made is that of the difference between the distribution of labelled and unlabelled code lines. In Figure 4-2, the distribution of the classification for lines with labels and lines without labels is plotted in a bar plot. The results for step level drawn are again representative for task and method level too. The one thing that sticks out is the relative growth of the unlabelled category. Giving the idea that if the model encounters a line that is not clear to it, the line is classified as unlabelled. The other classes more or less hold their relative distribution. From which nothing is derived for sure, but it strengthens the idea that the labelled part of the dataset is representative for the entire dataset.
This chapter showed it is possible to classify script lines with respect to the adapted process model. A supervised classifier has been trained that can annotate unknown lines of code reasonably well. The elements that served as input for the classification algorithms are Word2Vec features and labels from the adapted process model. These labels are manually assigned to Python function calls in code lines. This methodology needs two assumptions. First, only function calls in a given line of code are the decisive factor for determining the line’s classification in the adapted process model, all other Python language elements may be ignored for classification purposes. Second, Python code lines which are similar to each other in the Word2Vec embedding space have similar semantics, and thus should have similar labels with respect to the adapted process model. Both assumptions seem to hold within the experiments. The best results are reached with the Python2Vec model in combination with a Neural Network Classifier. This configuration is therefore used to annotate all lines of code in the next chapter, resulting in a line-by-line annotation for each script. These annotations represent the workflows followed in the scripts. In the next chapter, the collection of these workflows are further analysed.
Table 4-6: Mapping of the (mis-) classified labels with the Neural Network on Method level.

The Rows are the correct labels for the code line. The Columns are the Assigned labels by the Algorithm.

<table>
<thead>
<tr>
<th>True label</th>
<th>Calculated labels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Method 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28</td>
</tr>
</tbody>
</table>
5 Patterns and Insights in Data mining scripts

In the previous Chapter, classifying the lines of code in the python scripts with the concepts from the adapted KDD model gave a collection containing lists of labels. Each list represents the workflow of the script described with labels from the adapted KDD process model. In this Chapter, descriptive analyses on this collection of workflows show patterns and insights of general workflows in the data mining process. In this chapter the third research question will be answered, which is:

- **RC3:** Having a collection of scripts, with the code lines in the scripts annotated with concepts from the adapted KDD process model, which patterns and insights can be discovered?

The annotated dataset which was the end product from the previous Chapter is the subject of study in this Chapter. By means of different descriptive analysis techniques, the dataset is explored. First, the dataset is analysed looking for the existence of code blocks. Second, the co-occurrence of different labels will be analysed. Third, this is taken one step further, in which transitions from one label to another are studied. In this analysis, a transition model will be applied to the data, showing the transitions on different levels. This will give the first patterns and workflows used by Data-Scientists to solve their challenges.

4.1 Code Blocks

In this section, firstly block length will be analysed. Block length is defined as the number of consecutive lines that are assigned the same label. Code blocks are interesting because they give a more natural representation of a task then a single code line. Because a task is not programmed in a single line of code. More often a couple of lines are needed to complete tasks. While it was not possible to grasp a code comparing the block length with different filters makes clear whether filtering is needed to work with code blocks. Furthermore, comparing the block length between different levels makes clear how applying the different levels affects the number of code lines with the same label.

The first comparison of block length is that of unfiltered block length versus filtered block length. Filtered in this context is defined as skipping the categories “Unlabelled” and “Pre-processing” when counting the block length. The first expectation is that these blocks of code are interfered by the lines of code with Unlabelled and Pre-processing Labels. The second expectation is that programmers use two to six code lines in a block of code with the same label.

The first expectation on block length is examined by Figure 5-1, showing two histograms of the unfiltered and filtered block length at Step level. Figure 5-1 shows a shift from block length 1 to larger block lengths. This is expected. Hence, it is not remarkable on its own. That is to say, taking two of six categories out, the change of a random pick, being of the same category will grow. The remarkable element of the plot, however, is the proportional growth of the block length 3 and 4. This growth is more than expected and shows that by removing “Unlabelled” and “Pre-processing” lines of code, there is also a more than the substantial shift from small blocks to larger blocks of code. This proofs that Data-Scientists often need 2 or more lines of code to program a code block. Still, the code block is an aggregation of one-liners, blocks of code with multiple functions and misclassified lines.
The second element of block length that is examined is the difference between the three label levels in block length. This is interesting because now it is possible to strengthen the evidence that there is a notice of blocks of code. The three levels, of Step, Task and Method consist after filtered of, 4, 13 and 26 labels when there was no notice of blocks of labels. The expectation here is that on the more fine-grained level of labelling, the block length would reduce to a minimum. The Method level has 26 different labels making the change on an identical label is less than 4 percent. However, Figure 5-2, shows that on more fine-grained levels, the block length is decreasing, but not as much as would be expected if there was no notice of blocks. For example, the random change of a block length of three consecutive lines of code on Method level would be \( \left( \frac{1}{26} \right)^4 \approx 0.000002 \), in the set of annotated scripts this is around 0.05. This is three factors more, which shows there is a notice of code blocks in the results.

With this notice, the analysis this chapter continues. Lines of code with the same label will be treated as one block in the next sections. Also visible in the results is that “Unlabelled” and “Pre-processing” are acting as noise in code blocks and in between code blocks. Therefore, when applicable, these labels will be filtered out of the results.
4.2 Co-occurrence

After the confirmation in the previous section that consecutive lines of scripts with identical labels can be viewed as blocs of code, the next object studied is the observed co-occurrence of blocks compared with the expected co-occurrence. If two sort of blocks occurs more often together than expected, it will probably be because the one follows logically after the other, or the other way around. That is to say, in the latter case you first have to perform a certain task, before you can execute the task you want to execute. The expectation here is that different Tasks and Methods are related and this can be visualized by plotting the data.

The co-occurrence between block of labels in the code is visualized in a three matrix plot on three different label levels. By plotting the difference between the measured co-occurrence and the expected co-occurrence, these matrices show that two labels are occurring less or more than expected. The measured co-occurrence is the difference between the fraction of the scripts where Step A occurs in all the scripts where B occurs and the fraction of Step A in the total number of scripts. The selection of scripts is cleaned in advance, by removing all scripts with 3 labels or fewer. Resulting in 5675 remaining scripts.

Figure 5-3, Figure 5-4 and Figure 5-5 show the calculated co-occurrence matrices on Step, Task and Method level. First, some general observation will be discussed. After that, some more specific observation on Task and Method level will be given.

Starting with the least coloured rows first, it becomes clear that “Unlabelled”, “Selection” and “Pre-processing” have no other Categories that are co-occurring more often than expected. This is visible from Step Level to Task and from Step Level to Method level. On Method level, the “Pre-processing” is further divided into smaller parts and the non-relatedness can be assigned to the Method which belongs to the Task “Data-Processing”. An exception must be made for “Cleaning”, which has other Methods that co-occur more often and less often than expected.

The second remarkable finding is that the scripts with transformation elements are highly dependent on “Data mining”. The opposite is also true when a script contains a “Data mining” label, there is also a higher change on a “Transformation”. Next, scripts with a data mining step also have a higher change then average to contain an Interpretation step. Overall there is a higher change on co-
occurrence of “Transformation”, “Data mining” and “Interpretation”. This finding is different because the opposite is found in relation to other Steps.

After the high-level patterns are made visual, it is possible to zoom in on the Task level. Starting with “Pre-processing – Data sampling”, which is co-occurring with all “Transformation” and “Data mining” Tasks with an exception of “Clustering”. Other columns that are notable are “Classification” and “Ensemble”, both classes are more than expected related to the other “Transformation” and “Data mining” Tasks. This is something already visible on Step level. The Last column with the highest score is that of “Interpretation – Evaluation” clearly appearing more after “Ensemble” and “Regression” Tasks.

Zooming to the Method level, the same patterns as visible on task level appear with more detail. Some rows stand out, such as “Cleaning”, “Singular Value Decomposition (SVD)”, “Standardization” and “Combining”, These Method all have their own highly co-occurring and therefore related Method. For example, “Combining” is often co-occurring with “Logistic Regression”, this makes sense when one bears in mind that “Combining” is mostly done with more simple classifiers as the ones categorized under “Logistic Regression”.

The co-occurrence analysis shows that there are patterns in the data. Some categories do co-occur remarkably more with other categories. The patterns that become visible are logical and make sense if combined with knowledge from the Data-Science expert. In the next section will further explore the patterns found in co-occurrence.
3.4 Transition state analysis

This section analyses the transition states between the different classes of the adapted process model. After analysing the co-occurrence between different blocks of code in the previous section, this section goes a step further to see if there are insights in the transitions from one labelled code block to another. This is useful because the transitions analysis shows the patterns Data-Scientist use in their scripts. A future application would be to use these patterns to predict the next state or label when a script is in a certain state.

The assumption is that when modelling the collection of annotated scripts into transition state, patterns become visible and these patterns show how Data-Scientist build up their scripts. A transition state diagram or matrix models the change from one specific state A to another specific state B, when in a certain state A. We model this change as the fraction of transition from code blocks with label A to code blocks with label B, with respect to all transitions out of code blocks with label A.

Figure 5-5: Co-occurrence on Method Level
There are two possibilities to define the state in the data set. A first possibility is to see a line of code as a state. This first model defines the label of line 1 to line 2 as a transition. This is the analysis we start with on Step level, to confirm the findings from Section 4.1 on Block length. After that, the section continues with the second possibility. The second possibility is to take a block of code as a state. Filtering out self-loops, which are present in the first model, gives a much clearer view on the transition. For a clear image, some cleaning and pre-processing are executed on the dataset. The first noise elements taken out are the code lines, which are annotated in the “Unlabelled” Category. Second, all the scripts with 3 or fewer labels are filtered out. These are small scripts with, most of the time, only “Selection” and “Pre-Processing” lines. Taking them out cleaned the dataset from a lot of scripts that were not finished. This cleaning process resulted in a dataset of a little under 6000 scripts.

At Step level, the first model uses the first possibility by defining every annotated line of code as a state. This makes it possible to identify self-loops. In Figure 5-6 and Table 5-1 this model is visualized and as expected, there is a large change between 0.38 and 0.74. Here, a line is followed up by a line with an identical label, depending on the Step. This is confirming Figure 5-1, in which a lot of blocks of code lines with identical labels were present. Another element that is in line with earlier observations is the large transitions to “Pre-processing”. Therefore, first the loops, and after that “Pre-Processing” are filtered out to get a better view on the other classes.

![Figure 5-6: State transition on Step level, without filters](image-url)
The next step is to take out the loops in the state diagram, only taking into account state transitions between different labels. This new model is visualized in Figure 5-7 and Table 5-2. The first steps are very clear in this model, starting with “Selection” and followed up by “Pre-processing”. However, after these steps, the Figure is getting a little cluttered because all Steps have a strong connection with “Pre-processing”. This results in a lot of back loops. If “Pre-processing” taken out, as visualized in Figure 5-8 and Figure 5-9, a general workflow appears from “Transformation” to “Data mining” to “Interpretation” to “finish’. This is a pattern that is coming back on all the levels. Also, this is the pattern that confirms the high level conceptual model of Fayyad(Fayyad, Piatetsky-shapiro, & Smyth, 1996).
Table 5-2: State transition on Step level, without self-loops

<table>
<thead>
<tr>
<th>From</th>
<th>Selection</th>
<th>Pre-processing</th>
<th>Transformation</th>
<th>Data mining</th>
<th>Interpretation</th>
<th>Finish</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>0.59</td>
<td>0.27</td>
<td>0.04</td>
<td>0.04</td>
<td>0.06</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Selection</td>
<td>0.71</td>
<td>0.07</td>
<td>0.05</td>
<td>0.13</td>
<td>0.04</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Pre-processing</td>
<td>0.20</td>
<td>0.21</td>
<td>0.26</td>
<td>0.26</td>
<td>0.07</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Transformation</td>
<td>0.04</td>
<td>0.63</td>
<td>0.18</td>
<td>0.10</td>
<td>0.05</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Data mining</td>
<td>0.06</td>
<td>0.49</td>
<td>0.07</td>
<td>0.32</td>
<td>0.06</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Interpretation</td>
<td>0.09</td>
<td>0.47</td>
<td>0.05</td>
<td>0.12</td>
<td>0.26</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>0.99</td>
<td>2.57</td>
<td>0.43</td>
<td>0.66</td>
<td>0.87</td>
<td>0.48</td>
<td>6.00</td>
</tr>
</tbody>
</table>

Figure 5-8: Transition state diagram on Step level without Pre-processing

The elicitation of this workflow pattern took some rough cleaning and filtering. Two backward transition patterns towards “Pre-processing” and “Selection” are visible that are remarkable. The backloop to “Pre-processing” can be explained by regarding “Pre-processing” as all small data-manipulation needed in the scripts. These data manipulations are needed to enable the transition from one step to the next. These data manipulations are also implemented within a Step to pre-process the data for the next function call. That makes lines classified as “Pre-processing” a large source of clutter, which hides patterns between other states. Therefore “Pre-processing” is taken out in the analysis of the lower levels. The other back loop towards “Selection” can be subscribed to three causes. First, after a certain step in the script, a new data source is introduced. This data source needs to be imported, which causes the step back to “Selection”. Second, to the end of the process, the new knowledge needs to be exported, normally this is done by “Interpretation IO”, but these are functions close to “Selection IO”, so these lines are misclassified. Third, importing data is
mostly done via the Pandas library. The Pandas library is an extensive library with also incorporates a lot of other, mostly “Pre-processing” function calls. Function calls from the same library have a certain similarity and therefore are mixed up or misclassified.

![Diagram](image)

*Figure 5-9: State transition on Step level, filtering out pre-processing*

The analysis is continued on the lower Task Level. As stated before, self-loops and “Pre-processing” are filtered out. The resulting diagrams are Figure 5-10 and Table 5-3. Where Figure 5-10 only shows the transition with a change higher than 0.1. Still, scripts start with a “Selection IO” Task. After that, the majority of the steps will continue to Pre-processing, but these Tasks are filtered. Now it can be observed that the next step after selection will probably be “Transformation Transformation”, “Data mining Classification” or “Interpretation IO”

Going a step further in the process: after a “Transformation” Task, the next task will probably be “Classification” or “Ensemble”. Another remarkable observation is that after “Transformation Extraction” there is often a block of “Visualisation”, which was also visual during the co-occurrence analysis.

After a “Data mining” task, the next task is probably a different “Data mining” task. Or it is followed by the “Interpretation” tasks of “Evaluation” or “IO”. The exception in “Data mining” step is “Clustering”, which is most often followed by a visualisation. Again, this was also noticed in the co-occurrence analysis.

The last remarkable pattern, but again very logical, is that the last task in the scripts is most often an “IO” operation, which results in an upload or download operation towards the server or disk.
Figure 5-10 State transition on Task level

After Task level, the latest and most fine-grained level to analyse is the Method level. With 32 classes, the diagram and matrix get more disturbed. However, Figure 5-11 and Table 5-4 show interesting patterns, but these are on a more detailed level. Again, “Pre-processing” Methods and self-loops are filtered out. In the Alluvial diagram, only the transition factors higher than 0.1 are drawn.

Starting with a confirmation of the biggest pattern on Task level, again, after methods from the “Data mining” and “Transformation” step, the next task is “Evaluation” or “IO”. After “IO” most of the scripts are finished.

On the Method level, more detailed patterns are visible. “Data mining” and “Transformation” Methods are often followed by “Boosting”. Apparently, this is a popular and successful method in this repository of scripts, especially in combination with other methods. This finding is not obvious from the more global analysis.

Another fine-grained pattern is that after any other “Transformation” Method, there is a reasonable change because it is followed by a “Transformation” applying a “Normalisation”. So, Transformations do not stand for themselves. Something else that is remarkable in the matrix is the relation between “Singular Value Decomposition” (SVD) and “Evaluation”. This gives the notion that an “SVD” probably needs to be evaluated to measure the resulting transformation.

Furthermore, there are also a lot of individual relations that becomes visible. An example is a relation between “KNN” and “Random Forest”. These are not directly related, but a preliminary assumption is that if a data-scientist is testing general classification algorithms on his data, “KNN” and “Random Forest” are the easy candidates with, generally, good results.

In the diagram, it is also visible that a script stops often after “PCA” or “Discretization”. Here there is not even an assumption but probably people stop their script if they tried these methods.
Altogether, there are very interesting patterns in the repository on Method Level. When lowering the threshold below the 0.1 value used here, other relations become visible too. But these are relations that need to be explored one by one to identify their usefulness.

Figure 5-11 State transition on Method level

4.4 Conclusion

This chapter described the analysis of the repository of annotated scripts. The analysis gave an initial understanding how the concepts of the adapted KDD process model are applied in the scripts. The analysis of the data showed some interesting patterns. The first analysis confirmed the assumption that the distinct labelled lines of code can be aggregated to blocks of code with clearly definable labels. The code lines in these blocks of code are part of the same Step, Task or Method. However, in the raw data, these blocks are dirty and difficult to distinguish by “Pre-processing” and “Unlabelled” lines of code that are part of the block but not labelled as such. Therefor code lines with unlabelled and Pre-processing labels are filtered of, before further analysis. The second analysis shows that there are Tasks and Methods that are often used together. The interesting patterns found in this analysis is further explored in a Transition State model. This transition state analysis between the different blocks of code showed that there are commonly used workflows in the data. On the highest step level, the Adapted KDD Process model of Fayyad appears as the major pattern. On the lower level, more detailed patterns and connections are found. These patterns give a better understanding how Data Scientist build up their different workflows.

This experiment and dataset make it is possible to scout for interesting patterns. However, the applied analysis is only a first attempt to discover these patterns. Follow up research has to prove whether and how these patterns can be generalized and used in the support of other Data Scientist.
Table 5-3: Transition matrix, from one Task to another task, without loops, without Pre-processing

<table>
<thead>
<tr>
<th>Labels</th>
<th>Selection_IO</th>
<th>Transformation_Extraction</th>
<th>Transformation_Selection</th>
<th>Transformation_Transformation</th>
<th>Data mining_Classification</th>
<th>Data mining_Clustering</th>
<th>Data mining_Ensemble</th>
<th>Data mining_Regression</th>
<th>Interpretation_Evaluation</th>
<th>Interpretation_Visualization</th>
<th>Interpretation_IO</th>
<th>finish</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>0.67</td>
<td>0.04</td>
<td>0.01</td>
<td>0.04</td>
<td>0.05</td>
<td>0.00</td>
<td>0.01</td>
<td>0.02</td>
<td>0.03</td>
<td>0.07</td>
<td>0.05</td>
<td>0.01</td>
<td>1.00</td>
</tr>
<tr>
<td>Selection_IO</td>
<td>0.06</td>
<td>0.05</td>
<td>0.21</td>
<td>0.11</td>
<td>0.01</td>
<td>0.05</td>
<td>0.03</td>
<td>0.06</td>
<td>0.09</td>
<td>0.18</td>
<td>0.16</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Transformation_Extraction</td>
<td>0.13</td>
<td>0.06</td>
<td>0.14</td>
<td>0.12</td>
<td>0.01</td>
<td>0.08</td>
<td>0.05</td>
<td>0.19</td>
<td>0.11</td>
<td>0.04</td>
<td>0.06</td>
<td>0.11</td>
<td>1.00</td>
</tr>
<tr>
<td>Transformation_Selection</td>
<td>0.19</td>
<td>0.10</td>
<td>0.11</td>
<td>0.12</td>
<td>0.01</td>
<td>0.12</td>
<td>0.02</td>
<td>0.09</td>
<td>0.15</td>
<td>0.02</td>
<td>0.07</td>
<td>0.11</td>
<td>1.00</td>
</tr>
<tr>
<td>Transformation_Transformation</td>
<td>0.16</td>
<td>0.08</td>
<td>0.03</td>
<td>0.24</td>
<td>0.00</td>
<td>0.10</td>
<td>0.06</td>
<td>0.09</td>
<td>0.06</td>
<td>0.06</td>
<td>0.11</td>
<td>0.11</td>
<td>1.00</td>
</tr>
<tr>
<td>Data mining_Classification</td>
<td>0.11</td>
<td>0.03</td>
<td>0.02</td>
<td>0.09</td>
<td>0.00</td>
<td>0.21</td>
<td>0.05</td>
<td>0.18</td>
<td>0.03</td>
<td>0.21</td>
<td>0.08</td>
<td>0.10</td>
<td>1.00</td>
</tr>
<tr>
<td>Data mining_Clustering</td>
<td>0.11</td>
<td>0.10</td>
<td>0.03</td>
<td>0.08</td>
<td>0.12</td>
<td>0.01</td>
<td>0.02</td>
<td>0.04</td>
<td>0.19</td>
<td>0.19</td>
<td>0.13</td>
<td>0.13</td>
<td>1.00</td>
</tr>
<tr>
<td>Data mining_Ensemble</td>
<td>0.10</td>
<td>0.01</td>
<td>0.01</td>
<td>0.06</td>
<td>0.29</td>
<td>0.00</td>
<td>0.07</td>
<td>0.22</td>
<td>0.04</td>
<td>0.13</td>
<td>0.06</td>
<td>0.10</td>
<td>1.00</td>
</tr>
<tr>
<td>Data mining_Regression</td>
<td>0.11</td>
<td>0.11</td>
<td>0.01</td>
<td>0.03</td>
<td>0.24</td>
<td>0.00</td>
<td>0.13</td>
<td>0.19</td>
<td>0.02</td>
<td>0.10</td>
<td>0.05</td>
<td>0.10</td>
<td>1.00</td>
</tr>
<tr>
<td>Interpretation_Evaluation</td>
<td>0.12</td>
<td>0.04</td>
<td>0.01</td>
<td>0.10</td>
<td>0.21</td>
<td>0.00</td>
<td>0.21</td>
<td>0.07</td>
<td>0.06</td>
<td>0.10</td>
<td>0.08</td>
<td>0.10</td>
<td>1.00</td>
</tr>
<tr>
<td>Interpretation_Visualization</td>
<td>0.17</td>
<td>0.10</td>
<td>0.05</td>
<td>0.04</td>
<td>0.07</td>
<td>0.01</td>
<td>0.17</td>
<td>0.01</td>
<td>0.05</td>
<td>0.17</td>
<td>0.17</td>
<td>0.17</td>
<td>1.00</td>
</tr>
<tr>
<td>Interpretation_IO</td>
<td>0.20</td>
<td>0.01</td>
<td>0.01</td>
<td>0.04</td>
<td>0.04</td>
<td>0.00</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.06</td>
<td>0.55</td>
<td>0.55</td>
<td>1.00</td>
</tr>
<tr>
<td>Total</td>
<td>2.06</td>
<td>0.68</td>
<td>0.27</td>
<td>0.94</td>
<td>1.62</td>
<td>0.05</td>
<td>1.12</td>
<td>0.43</td>
<td>1.19</td>
<td>0.89</td>
<td>1.24</td>
<td>1.54</td>
<td>12.00</td>
</tr>
</tbody>
</table>
Table 5-4: Transition matrix, from one Method to another Method, without loops, without Pre-processing

| Rijlabels | Transformation_Extraction_PCA | Transformation_Extraction_SVD | Transformation_Selection_Hypothesis test based | Transformation_Selection_Other | Transformation_Selection_Univariate | Transformation_Transformation_Normalization | Transformation_Transformation_Scale | Data-Mining_Classification_KNN | Data-Mining_Classification_LDA | Data-Mining_Classification_Logistic regression | Data-Mining_Classification_Neural Network | Data-Mining_Classification_Random-Forest | Data-Mining_Classification_SVR/SVM | Data-Mining_Regression_Linear | Data-Mining_Regression_Logistic regression | Data-Mining_Regression_Random Forest | Data-Mining_Regression_SVR/SVM | Data-Mining_Regression_Tree | Data-Mining_Regression_Partition model | Data-Mining_Ensemble_Combing | Data-Mining_Ensemble_Boosting | Data-Mining_Evaluation | Data-Mining_Implementation_Evaluation | Data-Mining_Implementation_Implementation | Data-Mining_Implementation_Implementation | Data-Mining_Implementation_Implementation | Eindtotaal |
|-----------|-------------------------------|-------------------------------|-----------------------------------------------|-------------------------------|-----------------------------------|-------------------------------------------|----------------------------------|-------------------------------|-------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|
| Selection_IO | 0.06 | 0.06 | 0.06 | 0.02 | 0.04 | 0.02 | 0.07 | 0.02 | 1.00 |
| Transformation_Extraction_Other | 0.03 | 0.03 | 0.13 | 0.04 | 0.02 | 0.05 | 0.06 | 0.06 | 0.19 | 0.16 | 1.00 |
| Transformation_Extraction_PCA | 0.11 | 0.13 | 0.06 | 0.05 | 0.06 | 0.04 | 0.04 | 0.04 | 0.09 | 0.12 | 0.07 | 0.06 | 1.00 |
| Transformation_Selection_Hypothesis test based | 0.14 | 0.07 | 0.04 | 0.05 | 0.05 | 0.03 | 0.11 | 0.07 | 0.04 | 0.08 | 0.04 | 0.17 | 1.00 |
| Transformation_Selection_Other | 0.03 | 0.04 | 0.02 | 0.07 | 0.06 | 0.02 | 0.05 | 0.02 | 0.20 | 0.06 | 0.05 | 0.05 | 1.00 |
| Transformation_Selection_Univariate | 0.15 | 0.02 | 0.31 | 0.06 | 0.02 | 0.06 | 0.04 | 0.02 | 0.03 | 0.10 | 0.03 | 0.07 | 1.00 |
| Transformation_Transformation_Discretization | 0.07 | 0.19 | 0.02 | 0.03 | 0.04 | 0.02 | 0.11 | 0.11 | 0.06 | 0.02 | 0.05 | 1.00 |
| Transformation_Transformation_Normalization | 0.10 | 0.04 | 0.04 | 0.07 | 0.04 | 0.04 | 0.04 | 0.14 | 0.05 | 0.03 | 0.03 | 0.09 | 1.00 |
| Transformation_Transformation_Scale | 0.11 | 0.04 | 0.03 | 0.18 | 0.11 | 0.07 | 0.06 | 0.05 | 0.04 | 0.05 | 0.07 | 0.03 | 1.00 |
| Transformation_Transformation_Scaling | 0.04 | 0.31 | 0.04 | 0.04 | 0.08 | 0.31 | 0.04 | 0.04 | 0.08 | 1.00 |
| Data-Mining_Classification_Bayes | 0.08 | 0.04 | 0.04 | 0.08 | 0.04 | 0.08 | 0.10 | 0.04 | 0.03 | 0.03 | 0.08 | 0.10 | 0.06 | 1.00 |
| Data-Mining_Classification_KNN | 0.11 | 0.04 | 0.14 | 0.11 | 0.04 | 0.05 | 0.03 | 0.08 | 0.10 | 0.05 | 0.07 | 0.08 | 1.00 |
| Data-Mining_Classification_LDA | 0.05 | 0.02 | 0.04 | 0.11 | 0.04 | 0.05 | 0.03 | 0.18 | 0.07 | 0.12 | 0.02 | 1.00 |
| Data-Mining_Classification_Logistic regression | 0.09 | 0.05 | 0.06 | 0.02 | 0.03 | 0.06 | 0.12 | 0.16 | 0.02 | 0.08 | 0.03 | 0.14 | 0.04 | 1.00 |
| Data-Mining_Classification_Neural Network | 0.05 | 0.03 | 0.12 | 0.15 | 0.02 | 0.17 | 0.03 | 0.10 | 0.02 | 0.13 | 0.03 | 1.00 |
| Data-Mining_Classification_Random-Forest | 0.16 | 0.03 | 0.04 | 0.03 | 0.03 | 0.03 | 0.21 | 0.03 | 0.14 | 0.02 | 0.21 | 0.04 | 1.00 |
| Data-Mining_Classification_SVR/SVM | 0.12 | 0.03 | 0.04 | 0.13 | 0.13 | 0.03 | 0.12 | 0.14 | 0.09 | 1.00 |
| Data-Mining_Classification_Tree | 0.04 | 0.03 | 0.05 | 0.14 | 0.15 | 0.07 | 0.05 | 0.05 | 0.14 | 0.10 | 1.00 |
| Data-Mining_Clustering_Partition model | 0.06 | 0.09 | 0.10 | 0.07 | 0.08 | 0.04 | 0.02 | 0.03 | 0.02 | 0.18 | 0.09 | 0.07 | 1.00 |
| Data-Mining_Ensemble_Combing | 0.30 | 0.08 | 0.12 | 0.07 | 0.10 | 0.04 | 0.16 | 0.04 | 0.16 | 0.04 | 1.00 |
| Data-Mining_Ensemble_Boosting | 0.05 | 0.15 | 0.05 | 0.05 | 0.10 | 0.10 | 0.10 | 0.10 | 0.10 | 0.10 | 0.10 | 1.00 |
| Data-Mining_Regression_Linear | 0.05 | 0.02 | 0.04 | 0.05 | 0.08 | 0.01 | 0.14 | 0.04 | 0.21 | 0.03 | 0.09 | 0.07 | 1.00 |
| Data-Mining_Regression_Logistic regression | 0.10 | 0.10 | 0.07 | 0.03 | 0.06 | 0.03 | 0.22 | 0.10 | 0.03 | 0.10 | 0.04 | 1.00 |
| Data-Mining_Regression_Random Forest | 0.13 | 0.06 | 0.04 | 0.02 | 0.02 | 0.04 | 0.21 | 0.21 | 0.12 | 0.08 | 0.08 | 1.00 |
| Data-Mining_Regression_SVR/SVM | 0.19 | 0.06 | 0.02 | 0.03 | 0.08 | 0.06 | 0.21 | 0.21 | 0.12 | 0.08 | 0.08 | 1.00 |
| Data-Mining_Regression_Tree | 0.16 | 0.03 | 0.04 | 0.02 | 0.03 | 0.08 | 0.54 | 1.00 |

Eindtotaal: 2.81, 0.83, 0.29, 0.14, 0.08, 0.18, 0.31, 0.24, 2.36, 0.90, 0.02, 0.22, 0.47, 0.07, 1.36, 0.26, 1.68, 1.20, 0.39, 0.15, 2.90, 0.04, 0.36, 0.77, 3.51, 1.55, 2.38, 2.53, 28.00
Discussion and Future work

Adapted KDD process model

The Adapted KDD process model is the newly developed model to describe scripts. Data mining scripts and the conceptual process steps of the model are both data-oriented. This enables the model to connect the conceptual process with the concrete implementations of the Data mining scripts. The connection is made by splitting the conceptual steps into tasks and subsequently splitting these tasks into methods. This structure, from generic to specific, has consequences for the applicability of the model to other domains and applications. On Step Level, the model is conceptual and applicable on any domain. However, at Method level, choices have been made to match the model with the application. This has implications when the methods are transferred to a different application. The Methods from the adapted KDD model will need some adjustments when applying the model on another application. Depending on the new application, some methods will be less popular and are therefore not supported in the new dataset and therefore will fit better under a collective label. Other Methods, which do currently not appear enough in the dataset to justify their own label, could be popular in other new applications, which makes it possible to describe them with a separate label.

The tuning and adjustment of the model to the used dataset can be illustrated by the clustering task. In the current model, one of the Data Mining Tasks is Clustering. Clustering is often used to describe data, by grouping data instances into clusters based on the features of these instances. The literature on clustering describes many methods to calculate clusters. However, our data set showed that only one method is used. This method is the partition model. Therefore, the Adapted KDD process model only has two Methods under the Task Clustering; Partition Model and Others. This makes the model widely applicable because all Clustering implementations are described, as Partition model or as a different model. It would, however, be better to examine whether additional cluster methods could or should be added when applying the model to a different dataset.

This example on Clustering is representative for multiple tasks in the model. The Python scripts on Kaggle are focussed on predictive modelling. Classification methods are therefore well addressed in the model. For clustering the opposite is true. Clustering is a descriptive task and does not occur that often. Several tasks are not specified into Methods. The best example of such a task is Visualization. An analysis of the used visualizations in Data mining would be interesting. Unfortunately, there were too few examples of visualization in the dataset to justify a further breakdown into Methods.

Not only the domain and application are sources of other methods. Time is also a factor of importance. Data Science is a dynamic research field, where new techniques are invented regularly. For example, t-SNE was not invented five years ago. Now it is a popular and successful Method to perform dimension reduction and visualize high-dimension datasets (van der Maaten, 2014). Concluding, every time the model is applied again, verification of the model is recommended. This makes sure that the model stays aligned with the state of the art. Furthermore, it keeps the model aligned with the implementation level.

Semantic similarity of code

During the classification of code lines, semantic similarity is used to transfer labels from classified to unclassified code lines. This is proven to be a successful way to apply the labels to the full dataset. The Word2Vec algorithm is used for extracting feature vectors from the code, which describes this semantic. There have been experiments with Word2Vec on Python code. However, to the best of my knowledge, this is the first time it is used as an element of a useful application.
During the project, I experimented with several ways to employ Word2Vec. Eventually, two feature models have reached the report; the first model trained on the relatively small but targeted Python codebase. The second model is trained on a large but external source, the Question and Answer Forum StackOverflow. Test results show that the best results were achieved with the first model, called the Python2Vec model. This Python2Vec model is therefore used for the rest of the analysis. The Word2Vec model has proven itself as a useful algorithm for describing code, but it could still use further research. Although it is empirically proven to work as intended, the theoretical background is an open research challenge in the NLP research field. Further optimization and understanding of Word2Vec feature extraction would allow a better application of this technique. This may include alternative ways to highlight code base repositories, for example by finding similar pieces of code and making workflows clearer.

**Code blocks**

The analysis of the data shows the existence of code blocks. Code blocks are consecutive lines of code, with the same label. This is a very strict definition of a code block, making the identification of a block highly dependent on the correct classification of the code lines. The use of code blocks makes it necessary to use firm filtering measures before the further analysis of co-occurrence and state transition can be performed. Furthermore, even after filtering, the percentage of single line blocks is 38%, which is higher than the expected 20%. This gives an indication that there is still much noise in the data. The current definition of code blocks follows the heuristic to classify single line of code independently. Future work could extend the followed method by developing a more comprehensive method to identify and classify code blocks.

From the experience of this research, two ways forward are suggested. First, another heuristic could be taken on the processing of code, by processing code blocks instead of code lines. Code blocks could, for example, be defined as separated by comments, separated by blank lines or by the blocks in a Notebook. However, this implies more restrictions on the studied scripts, because the free formatted, raw and unorganized collection of Python scripts of this study makes it impossible to take this approach. A second way forward is to combine multi-label classification with an extra combinational classifier. Multi-label classification gives a probability score for multiple labels. Where a standard classifier would simply assign the label with the highest probability, the combinational classifier used the probability of the multi-label classifier to calculate a smoother label. Multi-label classification was too advanced for the developed process used in this project. However, future work on code classification could study the use of multi-label classification as input for a classifier that also takes into account the classification of lines before and after a target script line.

**Co-occurrence**

In the co-occurrence analysis, old and new knowledge is extracted from the data. The relationship and non-relationship between different Steps, Tasks and Methods are visualized. These visualizations show trends about the most popular combinations of Tasks and Methods. From a Data-Science background, some of the co-occurring Tasks and Methods follow expected patterns. For example, the relationship between “Clustering” and “Visualization” or “Classification three” and “Train, test, split” will not come as a surprise for the intermediate Data-Scientist. Other combinations are less obvious, for example, the relationship between “Data Processing”, “Selection” and “Visualization” or the relationship between “Standardization” and “Normalization”. Both expected and unexpected results are interesting. The expected patterns are interesting because they confirm existing knowledge and can be used directly in a follow-up experiment. The less expected
results are interesting because they call for further study and experiments. Perhaps they are just random noise or they are the first signs of new knowledge extracted from the data.

The different co-occurrence plot show relatedness on a continuous scale. In this project, there was no goal to define a measure for significant co-occurrence, by setting a threshold to co-occurrence at a certain value. However, the next step could be to find such a threshold for relatedness. This would make it possible to decide which other Tasks or Methods could be recommended when encountering a Method in a script. However, finding such a threshold will be challenging, because relatedness depends on many factors. A possibility is to find such a threshold with the help of a group of experts, but perhaps the State Transition diagrams could be used as well.

State Transition

The State transition analysis is the last and most important analysis of this project. In this analysis, patterns are extracted from the data, adding information to the existing knowledge. The State transition analysis is executed on the three different levels of the model. The observation on these three models and between these three levels all give other and additional information. The results will be discussed, starting on the high conceptual Step level, continuing on more detailed Method level that is very close to the implementation. To end with some general remarks on the state transition model

The first interesting observation on Step level is the confirmation of the Data-outline of Fayyad’s KDD process model. The State transition on Step level, displayed in Table 5.4 and Figure 5-10, clearly shows a forward transition over the steps from start to finish, which matches the steps from Fayyad. To show these patterns, some firm filtering had to be applied, this will be discussed later. After this filtering, the numbers show that after a line of code with a certain label, the next line has the largest chance to get an identical label, staying in the same Code block. The second possibility is that the next line has the label of the next step. This confirms the conceptual model and makes it possible to use this conceptual model to give recommendations for the next step.

The second interesting pattern on Step level is the backward pattern. This is most clearly visualized in Figure 5.9, showing two backward patterns towards Selection and Pre-processing. From “Transformation”, “Data mining’ and “Interpretation”, a substantial part of the transitions go back to “Selection” and “Pre-processing”. There are two things to learn from the backward pattern. First, the backward transition between steps is implicitly part of the conceptual KDD process model, now this pattern becomes visual in real applications. Second, the scripts do not follow a linear order but go back to earlier steps and these backward steps can be quantified. All together this backward pattern should be part of the further analysis and of any application.

Before the forward and backward patterns became clear, some heavy filters in the data had to be applied. These two filters also tell something about the data. The first filter is to remove self-loops from the state by changing the definition of the state, from lines of code to identical blocks of code. This makes it impossible to change to the same state, highlighting the transition between code blocks. Second, the Pre-processing labels are filtered out. Pre-processing appeared to be too dominant as a label and cluttered all other patterns. The label Pre-processing is appearing everywhere in the scripts, before and after any other label. The reason becomes clear when examining the function calls which are assigned the label “Pre-processing”. These function calls are a mix of all the different implementations of data manipulation. These data manipulations are the glue between one Task and the next, they are therefore used everywhere in the scripts. This makes them part of the other blocks of code which are not labelled “Pre-processing”. Therefore, in this set-up,
is opportune to filter them out in order to get better results. In future work, it would be better to use a stricter definition for Pre-processing. Future work will also benefit from relabelling the function calls that are used to perform data manipulations in and between the code blocks to Unlabelled.

The most detailed patterns are visible on the Method level. Besides the global patterns that confirm the patterns on Task level, there are also specific one to one relationships visible, where an individual Method is followed most of the time by one other individual Method. An example is that after “Bayes Classification”, there is a probable transition to “Random Forest”. This confirms the relationship already found in the Co-occurrence analysis. These types of links can provide a very specific recommendation about the next method to implement.

More general patterns are also visible. For example, after applying a Data mining method, there will likely follow another Data mining method, with a high chance to be “Ensemble Boosting”. After finishing the “Data mining”, the next Task will be “Evaluation”.

Another very strong and remarkable relationship noticeable in the plots is that after a random classification method, there is a considerable factor of 15 percent, which is followed by a “Classification Ensemble” method. Considering that most of the scripts are written to perform classification tasks and to make as accurate predictions as possible. This seems to be a very interesting option when the goal is to boost your prediction scores.

There are also some patterns on Method level which go back to a previous step in the process. For example, after “Data mining - K Nearest Neighbour (KNN)”, in 14% of the cases, the scripts go to “Normalization”. This is unexpected because normalization is usually performed before “KNN”. There are two explanations for this pattern. Either scripts are not as linear as we expect and Data-scientists first program a “KNN” classifier function and afterwards perform “Normalization” and call this function. Or, people are less skilled than expected, so it would be helpful to give them advice on how to construct their Process.

The adapted KDD process model is not finished. In the last Step, which is an evaluation, there are no methods defined and therefore the classification stops at Task Level. This choice was made because the dataset did not have enough support to split up the Tasks and to train a classifier on them. However, it is interesting to know which “Evaluations” and “Visualizations” people use in “Data Mining”. If this is part of future work, it would be possible to extend the chain of Methods from “Pre-processing” to “Transformation” to “Data mining” with an “Evaluation” step. This would make it stronger, giving insights in more patterns and give better recommendations.

Modelling the dataset into state transitions gives very useful insights on how Data-Scientists follow up one Task or Method after the other. This is very useful to assist unexperienced data scientists on how to code their scripts. It would be interesting to look to longer workflows, to discover more complex patterns of multiple Tasks or Methods. However, this is easier on a cleaner dataset. In the next section, recommendations will be made on how to reach such a cleaner dataset.

**Future work**

There is a need for an improved dataset in order to get better and more advanced analysis results. This can be achieved by working on different elements from our classification pipeline. The first element that can be improved is the Transformation step. In the transformation step, a model called Python2Vec is made, which models semantic similarity between words in the Python code. This thesis shows that this Python2Vec model can be applied to its goal, taking the best practices from literature. The optimal settings for this Python2Vec model specifically and the Word2Vec algorithm,
in general, is still an open research question. The second element that can be approved, is that of classification. At this moment, two states of the art multi-nominal classification algorithms were selected, to classify every single line of code to one of the Methods. Further experiments could be performed with Multi-label classification, giving a probability for every line of code for every label. This would make it possible to extend the classifier with a combining classifier that takes into account output of the multi-label classifier for the target line of code, but also for the surrounding lines of code and possibly even other features. This could result in a more complex but informative classification that gives better insights into the blocks of code corresponding to the Methods, than the relative discreet method used now.

The developed adapted KDD process model has been a useful model to describe the programming scripts. It would be good to confirm this model and possibly extend it with use of a different repository of scripts. There are two options when choosing a new repository of data mining scripts. The first one is to shift from Python scripts to Python notebooks. The second one is to change programming language, for example, to study R scripts. In both cases, the model needs to be verified by selecting a representative collection of function calls of the programming language. In the case of Python notebooks, the expectation is that there will be more diversity in the Evaluation function calls because notebooks are more used for demonstration and exploration. This may create an opportunity to extend the model by assigning methods to the Evaluation and Visualization tasks. Another opportunity of changing to notebooks is to redefine the definition of code blocks. Notebooks are often better structured, giving the opportunity to experiment with different definitions. In the second case, when changing to R, or another language, a whole new mapping between function calls and methods from the model is needed. In R, it is more complicated to elicit a relevant set of function calls. The reason is that in R the calling of a function is possible in different ways. Furthermore, it is common practise to chain functions calls in one line, making it difficult to decide which one is decisive for the Method applied in the line. However, despite all these challenges, it would be very interesting to compare the current results with other script repositories.

When looking beyond data mining, the methods and techniques applied to classify and analyse scripts could be applied to non-data mining software repositories. Of course, a lot of work needs to be done again. First of all, a new model to describe the process of the target repository should be developed. Second, a mapping from function calls to this new model should be made. However, it could be very interesting to classify a repository using the techniques of semantic similarity and machine learning classification. It could show commonly used patterns, but also similar pieces of code or strange routines. This could all lead to a better understanding of the code repository, which is the first step to improvement.

The current results are the first step in the support of Data Scientists. A next step could be the development of an IDE or Browser plugin that gives recommendations to the Data Scientist, depending on what they have already written. The IDE could, for example, give a recommendation on the next Method. The programmer writes a block of code recognized as “Transformation – Extraction”. The IDE would propose to implement “Data mining – Classification” or “Data mining – Ensemble”, together with a list of proposed Methods: “SVM/SVC”, “Random Forest” or “Boosting”. Depending on the supported libraries, there could even be a suggestion on the function calls that are useful. Another option is to supply a reference to a script, where they apply such Methods. Coding is often searching for examples, supporting this search would be helpful. Finally, all the knowledge gained and extracted can be used to support future Data Scientists.
7 Conclusion and Outlook

In this chapter, an answer to the main research question: “To what extent can data mining programming scripts be described and classified by a KDD process model”, is provided. Additionally, an overview of the project contributions is given, as well as an outlook for future work.

7.1 Conclusion

In this work, the research on workflow patterns analysis in Data-Analysis programming scripts is described. Formalizing knowledge about these patterns is useful to improve and support the Knowledge Discovery and Data Mining Process.

The first part of the work is the formulation of an adapted KDD process model. This is an adapted model, which extends and aggregates existing models and ontologies found in literature. Furthermore, a relevant selection of the function calls used in Data Mining scripts are used. This adapted KDD process model is designed to align with the structure of the programming scripts. This makes the model useful for classification of scripts with a firm foundation in previous research.

The second part of the work describes the workflow used to classify the Python programming scripts from Kaggle with respect to the adapted KDD process model. This workflow follows an actual KDD process and makes it possible to annotate script lines with the Step, Task and Method labels from the adapted KDD model. After pre-processing, evolving selection, and cleaning of the scripts, there is a cleaned collection of script lines. The next Transformation step, extract features for every script lines, by training and applying two Python2Vec models. Python2Vec is an extension of the word-embedding algorithm Word2Vec. This is the first time that Python2Vec is applied as part of a workflow instead of just being the object of research. After the transformation step, the generated features are used as input for the Data-mining Step. In the Data-mining step, a different combination of models and classification algorithms were trained on the features and manually applied labels from the Adapted KDD process model. The Evaluation showed, that it is possible to extend known labels from labelled lines of code to unlabelled lines of code. The Evaluation also appointed the simplest Python2Vec model in combination with a Neural Network classifier as the best option for classification of the script lines. This combination is applied to assign all script lines a label from the Adapted KDD process model. Resulting in a collection of lists of labels per script, that represents the workflow of that script with respect to the adapted KDD process model.

The third part of the work is the analysis of the lists of labels, looking for patterns and insights on the data mining workflow. The first analysis on code blocks showed that there is a lot of noise in the data. However, the analysis results confirmed the hypothesis that scripts are built of blocks of code. The second analysis showed patterns of strong co-occurrence between different Steps, Tasks, and Methods from the adapted KDD process model. These patterns were confirmed in the third analysis on State transitions. The analysis on State transitions gave the most valuable results. The first useful result was the confirmation of the conceptual KDD process because it is the dominating transition pattern over the conceptual Steps. Furthermore, there are more detailed workflows identified between different Tasks and Methods in the process. These identified patterns are the first step towards recommendations in the KDD process.

All parts together show that it is possible to describe and classify programming scripts with respect to the KDD process model. The patterns and insights in this collection of scripts are useful. Further exploration of this and other collection of scripts is useful to support Data Scientist in their Data Extraction challenges.
7.2 Outlook

This Master thesis project was the first attempt on modelling and classifying data mining programming scripts with respect to the KDD process. Of course, there are elements to improve and next steps to take. The author wants to propose the following ones.

Further extension and tuning of the adapted KDD process model is needed to provide better support for later applications. Especially on Method level, the model could be extended. This can be done by selecting another dataset. Suggestions of other datasets are Python Notebooks or R scripts.

During transformation from code to features, a trained Python2Vec model is used for calculation of the features. This is a promising, but very new technique. Further research on Python2Vec could give better guidelines on how to apply this technique. Furthermore, it probably will show where this technique could be applied successfully as well.

In this thesis, classification is performed by straightforward Multi-nominal classification. This results in easy to use, but results in noisy data. Using more advanced classification techniques, such as multi-label classification and a combined classifier, could remove a lot of noise. Therefore, these techniques could make the resulting analysis even more valuable.

The most interesting results in this project are achieved by analysing the state transitions. The confirmation of the higher level conceptual KDD model together with the patterns on the lower abstraction level of Task and Method, provide good arguments for further exploitation of the State Transition model.

This research is the first attempt to study programming scripts. It shows a promising result that gives an invitation for further research to the optimal methodology and exploration of the dataset. This could result in tools to assist the Data Scientist, such as a recommender plugin for an Integrated Development Environment, which can make a recommendation on the next code block, or even further in the future a model-driven approach to Data Science workflows.
Acknowledgment

Finishing a second Master at the age of 35 is not common. I would like to thank the Royal Netherlands Navy to give me this opportunity to study Computer Science.

During my thesis, I got a lot of support from my supervisors Christoph Lofi and Alessandro Bozzon. Alessandro at a little distance of the daily business. And Christoph always available on Chat and when time allowed for a direct chat.

Furthermore, I like to thank the Kappa team for our (ir) regular meeting, presenting each other our Thesis projects and discussing common problems.

From the start, it was known that writing would become a hurdle. I would like to thank my team of correctors and helpers, Vincent, Vincent, Sandra, Matthijs and Jeroen for reading and correcting my work.

Graduation is a lonely process. I am especially grateful for Bas and Alexander for teaming up. Discussing each other’s project on a daily basis and assisting when wanted.

At last, I want to thank my wife Marthe. She is always my support, especially the last year. And my son Christian, because, his smile at the end of the day, makes all the thesis troubles disappear.

Jan Zegers
Bibliography


Mikolov, T., & Com, T. G. (2014). Distributed Representations of Sentences and Documents, 32.


