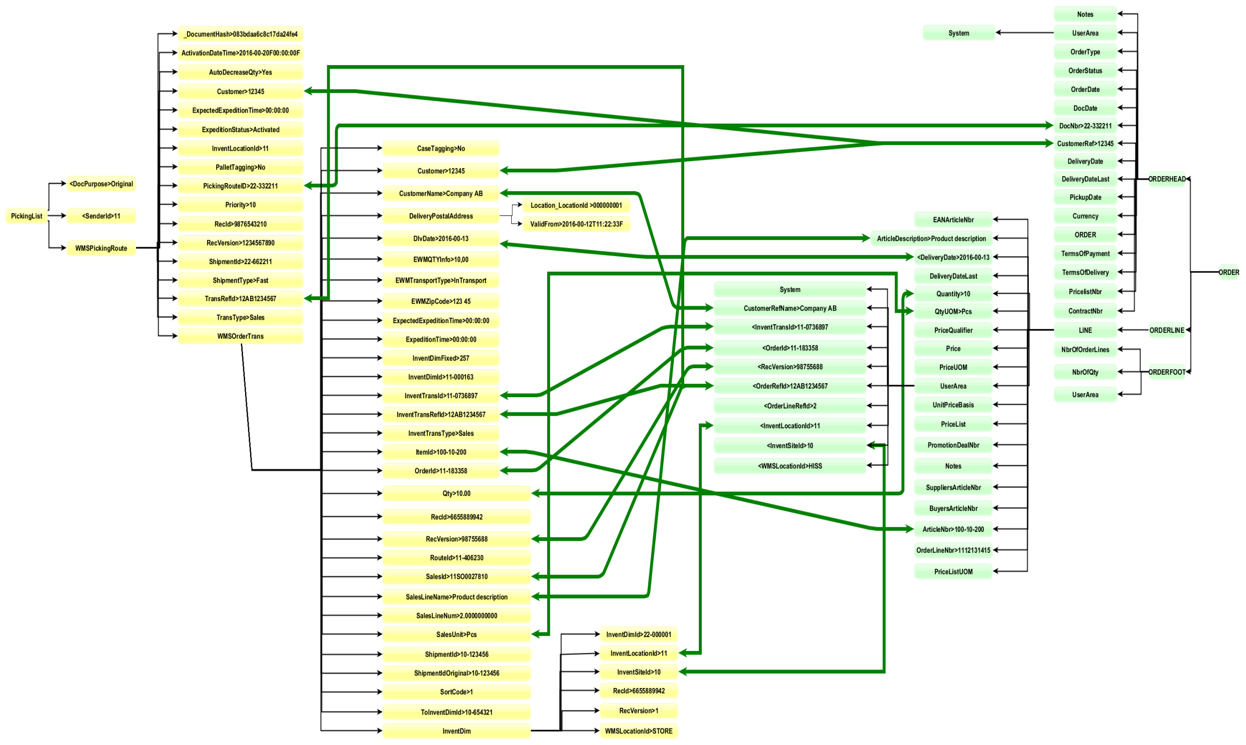
CHALMERS



Data integration using machine learning

Automation of data mapping using machine learning techniques

Master of Science Thesis Complex Adaptive Systems

MARCUS BIRGERSSON

GUSTAV HANSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Data integration using machine learning

Automation of data mapping using machine learning techniques

MARCUS BIRGERSSON GUSTAV HANSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2016

Data integration using machine learning
Automation of data mapping using machine learning techniques
MARCUS BIRGERSSON and GUSTAV HANSSON

Industrial Supervisors:	ULRIK FRANKE, OLOF WISTRAND AND MAGNUS FREDRIKSSON
Academic Supervisor:	CHIEN-CHUNG HUANG
Examiner:	CHRISTOS DIMITRAKAKIS

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A graphical illustration of a mapping example between two XML files. Illustrated by Marcus Birgersson

Typeset in LaTeX
Gothenburg, Sweden 2016

Data integration using machine learning
Automation of data mapping using machine learning techniques
MARCUS BIRGERSSON and GUSTAV HANSSON
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Data integration involves the process of mapping the flow of data between systems. This is a task usually performed manually and much time can be saved if some parts of this can be automated.
In this report three models based on statistics from earlier mapped systems is presented. The purpose of these models is to aid an expert in the mapping process by supplying a first guess on how to map two systems. The models are limited to mappings between two XML-formats, where the path to a node carrying data usually is descriptive of its data content. The developed models are the following:

1. A shortest distance model based on the concept that two nodes that have been associated with a third node but not each other most likely have something to do with each other.
2. A network flow model, which connects words with similar semantic meaning to be able to associate the words in two connected XML paths with each other.
3. A data value model which connects data values to nodes based on similarities between the value and earlier seen data.

The results of the models agrees with expectations. The shortest distance model can only make suggestions based on XML-structures that are present in the training set supplied for the project. The network flow model has the advantage that it only needs to recognize parts of a path to map two nodes to each other, and even completely unfamiliar systems can be mapped if there are similarities between the two systems. Overall, the data value model performs the worst, but can make correct mappings in some cases when neither of the others can.

Keywords: artificial intelligence, machine learning, system integration, data mapping

Acknowledgements

Firstly we would like to thank our supervisor Chien-Chung Huang for listening and giving feedback to our ideas and our examiner Christos Dimitrakakis for valuable pointers during the project.
We also want to thank the project group at iCore Solutions: Magnus Fredriksson, Olof Wistrand, Carl-Adam Wachtmeister, and Ulrik Franke at the Swedish Institute of Computer Science, for their help which guided us through the project and without them this project would not have been possible.
Finally we want to give an extra thanks to our friends and families for all the support during our education, and especially Ann Nylund for helping with the proofreading of this report.

Marcus Birgersson and Gustav Hansson, Gothenburg, January 2016

Contents

List of Figures

List of Tables

1 Introduction

The need for integration of different systems is steadily increasing, much thanks to digitalization, globalization and the Internet of things. Many companies start off small with just a few systems which need to be connected to each other, but if peer-to-peer communication is used the number of connections will grow exponentially when new systems are included in the business, which will be the case when the companies grow. If any of these systems are replaced, multiple new connection specifications need to be created for the new system to be able to communicate with the others. In cases like these an integration process with Service Oriented Architecture (SOA) needs to be implemented to prevent the costs of maintaining the system from rising above the profit received by the system. Even if a company from the start uses integration tools and SOA, it is common that it in a later stage will buy other companies that should be used as affiliated companies, rather than stripped of their assets. In these cases the need for integration will arise since the systems of the new companies need to be able to communicate with the original company.
Many systems are starting to use algorithms in artificial intelligence and machine learning to be able to automate manual tasks and to increase productivity and effectiveness for different processes. These algorithms are today used in many areas such as self-driving cars, robotics, search engines, spam filters and so on. Many tasks that was once thought too hard for a computer to perform are now reality. Since it is only a question of time until the integration market will be the bottleneck for faster increase in productivity for various companies, there is an urgent need to implement these tools in the integration market. The purpose of this project is to find ways to use tools from artificial intelligence and machine learning to ease the integration of information systems.

1.1 Background

Data integration is the process of mapping the flow of data between two or more systems. When companies have many different systems, all in need of communicating with each other, and at the same time strive to be able to replace any of the systems for another without needing to create new connection specifications to every other system, the demand for integration as well as Service Oriented Architecture arises. Service Oriented Architecture (SOA) is a design pattern used in many larger systems today. SOA is based on a number of principles, e.g.:

- The systems are considered to be consumers and/or producers of information

- The information is broken down into smaller reusable components
- Low dependencies between the systems simplify maintenance and interchange of parts of the system

To meet these demands for larger systems, advanced networks of mapping processes are set up to be able to handle version change of systems, connections between many to many systems etc. This is can be done by having a more or less static "intermediate" system, that all other systems is mapped towards, and all systems receive information from. By doing so, if one system changes, only the connection between the new system and the intermediate system needs to be updated, and all other systems will still work.

The process of setting up this system is partly manual, which is a time consuming process [7]. This process consists of taking the output from one system, identifying the relevant data for the intermediate system, and creating a mapping specification to move these data to the correct place in the intermediate system. The second part is to then do the same thing from the intermediate system toward all other systems that it needs to be able to communicate with. Since the integration market is growing, it is necessary to automate this process to prevent the integration process from being the bottleneck which limits companies to grow as fast as they could. By using algorithms from artificial intelligence and machine learning this process could be faster if a first guess of the mapping specification could be made by a computer, based on information from earlier mapped systems. The work that remains for the user would then be to validate if the mapping works by testing the connected systems, and in some cases correct some of the proposed mappings.

1.2 Introduction to data integration

Data integration is the process of moving data from one system to another. Basically one has some system that should communicate with another system, but they are not directly compatible with each other. As a simple example, the first system produces a file containing some data, which should be injected into the other system. It is known what data the other system needs, but to insert the data in the correct places the first system needs to know where the data should be placed. This is where data integration comes in; to create a schema for what data in one system another system needs, and where this data should be put.

A common format for storing and communicating arbitrary data structures is via the XML standard, and the data integration process will then be to identify the XML paths in the input file, and where to put its data in an output file. For example one system could contain orders put by customers, and another system that handles the invoices. One would then need to create an output file from the order database, and the invoice system should then be able to take that information to create an invoice. In this case, one could have the customer name stored in the output from the order system as *ORDER/ORDERLINE/CustomerRef*, but the invoice system will read this information from the XML path *SYSTEM/LINE/Customer*. One will then need to create a schema that maps *ORDER/ORDERLINE/CustomerRef* to *SYSTEM/LINE/Customer* so that the two systems are able to communicate.

For human experts, it is often a question of identifying paths that seem to describe the same data in the two formats, and then map these paths. The mapping can then be tested and evaluated, and if information is missing or ends up in the wrong place, the schema can be modified until everything is correct. In most cases it is about finding semantic similarities between the paths, trying out a mapping and looking at the result to evaluate if it seems to be correct or not.

1.3 Problem formulation

Using already mapped data, the task is to develop an algorithm based on machine learning tools and statistics to make a first step towards automating the mapping process.
The goal is to have a software that given a new set of inputs and outputs can make a first guess of how to map the flow of data and report a confidence level of these mappings. The purpose of this is to reduce the time it takes to manually map the system.

1.4 Delimitations

Data can be represented in a large number of ways. In this project the main focus will be on mapping data represented by one XML-file to another.
Generic mappings, where different data values are combined or split in some way, will not be considered. Focus will instead be on the cases where one data value in the input format is mapped to none, one, or several places in the output format.
Information contained in the attributes of the XML paths will not be used. The reason for this is that the given training data in most cases do not have any attributes, and using them sometimes would probably not contribute that much to the overall result, and the specific results from this approach would therefore be hard to evaluate.
Finally, external information regarding words will not be used, i.e. words will be treated as categorical and words that have not been seen before will not be matched to known words semantically using for example string comparison algorithms or external databases containing word clusters. The reason why this is not included in the project is because of time constraints. To be able to find satisfactory comparisons one would need to implement multiple cases to handle abbreviations, both formal and non-formal (i.e. nmbr, nbr and qty), as well as camel case written sentences. Since some words that are similar still could have a different meaning, one would have to handle even that case.
The reason why the use of external lexicons is not considered to match new words using semantics, is foremost that the semantics in an XML tree is not necessary of the same type of semantics that one finds in a natural language. For example, the root element of an XML tree is describing the tree and the format of that document, more than the actual data connected to the leaf. This makes the semantic meaning of one root element more or less equivalent with another root element in the process of mapping, but the meaning of that root element in the context of a natural language could be completely different. For example, two root elements could be *ORDER* and *PickingList*, which would be treated the same in the process of mapping two paths,

but few would argue that their meaning would be almost the same in the context of a natural language.

1.5 Related work

This project is closely related to schema matching, and the basic idea behind some of our methods are based on discoveries in this field [1]. Machine learning techniques have been used for the purpose of data mapping, but mainly to provide an integrated overview of data from disparate sources, and not for mapping pairs of systems [6].

1.6 Outline

In Chapter 2 we will introduce the mathematical and theoretical concepts needed for this project, including definitions used in the project and our probability models and some basic graph theory. In Chapter 3 the methods for predicting a mapping will be introduced, including how given data is used to train and validate the models. In Chapter 4 we will present the simulations made to evaluate different parts of the models and draw conclusions regarding the effect of parameters and so on. Finally in Chapter 5 we will present the results of the model together with a discussion of the predictive power of the models

2
Theory

The theory for this project will include a more specific definition of what will be considered a mapping and which mappings that are allowed in the scope of the project. We will present the statistical models used for computing mapping probabilities between both words and paths, and also what will be denoted as a ”path”.
In addition we will be presenting which scoring methods used in this project to be able to evaluate the prediction models, and the classification definitions of faulty and correct mappings. A short introduction to graphs, shortest distances in graphs and maximum flow in graphs will be presented, as well as the definition of N-grams and the symmetrized version of the Kullback-Leibler-divergence, used for measure distance between distributions.

2.1 Definitions

The following section presents a number of mathematical definitions used to explain the model in further detail. It also presents concepts and computational models used in this project. Most concepts are standard and is only presented to underline how it will be used in this context, and in some cases to narrow down the model for the specific use in this project.

2.1.1 XML-files

XML, or *Extensible Markup Language*, is a standard used for encoding information in a way that is both human- and machine readable and is commonly used for representing arbitrary data structures.
An XML-file stores information in a tree-structure of nodes with possible data values connected to the leaf nodes. Each node can be identified by its XPath.
XPath is a tool used for querying the file for its information. A path to a node consists of its name and all of its ancestors names, separated by a slash (/). The term ”path” will further be used to describe the full XPath to a leaf node.
The data contained by the XML-file is bound to the paths in the file. We say that a data value is connected to a path, if the path contains a data value. A ”leaf node” is a node that doesn’t have any child nodes.

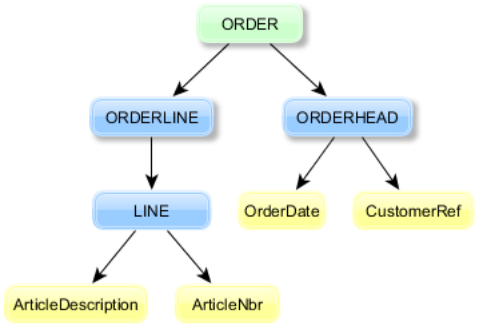


Figure 2.1: Example XML-tree describing an order. Root node in green and leaf nodes in yellow. The full path to the leftmost leaf node is "/ORDER/ORDERLINE/LINE/ArticleDescription"

2.1.2 Raw data

The raw training data consists of a number of previously mapped file pairs, i.e. two XML files containing the same data encoded in two different ways. Each file pair can be a mapping from either an input format to an intermediate format or an intermediate format to an output format. The intermediate format exists for practical reasons. In general it would be possible to just specify a mapping directly from one format to another, but these schema specifications would then end up being specific for these two systems. It is common that some system is supposed to map to several systems. If one maps A-B, A-C and A-D individually, all three specifications would need to be updated if the definition of A changes. A system built like this would be hard to reuse, and in addition direct mappings or hard connections in general result in cost and inefficient handling of change. If one instead specified the mappings A-I, I-B, I-C and I-D, one would only need to change one mapping specification if any of the systems changes, since the intermediate specification don't change.
The focus of this project is the mapping process between any two files, independently of if the file is an input, an output or an intermediate type, therefore each file pair is said to consist of an input format and an output format, where in some cases, the input format could be an intermediate format, and in some cases the output format could be an intermediate format.
Each file pair may originate from a number of different systems, for example, the formats could be from order systems, customer databases, invoice systems and so on. Even if two formats are from the same type of system, it could still differ much regarding how the XML trees are defined, since most systems generates their own XML files (based on the XML-standard but with their own choice of names for the tree nodes). These files will not necessarily be similar to one another. These file pairs will be denoted as files from different areas and it is assumed that a file pair from one area has little or nothing in common with a file pair from another area.
Since the available data is consisting of previously mapped files, it is important to notice that the mapping specification for these files are not known. The only available information is where the data values are located in the input format and where the

data values are located in the output format. An example of how a file pair could look like can be seen in Table 2.1. In the table one can see that the data value ”3”, connected to ORDER/LINE/qty, is unique in the input file, but not in the output file. In general, it is not possible to know if this path has been mapped towards List/Body/quantity, List/Body/lineNmr, both or none. If the integration procedure is unfamiliar and the natural language the paths are built from is unknown it is hard to know which paths that have been connected. These are the conditions that apply to the script parsing the file pairs of previous mapped data and generating mapping specifications using that information.
In the example from the table one can also see that other problems will arise when data values changes its format between the files. In this case, the company name and the date has made small changes which will make the algorithm treat them as completely different data, and therefore not conclude that they have been mapped. This problem is not as essential as the previous error, since this error ”hides” information from the models, but the previous error gives the models faulty information. Both errors will though still be a problem when validating a mapping suggested by the model.

Input document		**Output document**	
Input path	Data	Output path	Data
ORDER/LINE/DocNmr	12345	List/Body/DocHash	K23Hij&h
ORDER/LINE/qty	3	List/Body/quantity	3
ORDER/LINE/DlvDate	20160125	List/Body/lineNmr	3
ORDER/LINE/Customer	Company LTD	List/Body/DeliveryDate	160125
ORDER/LINE/Id	01234	List/Body/CustomerRef	01234
ORDER/LINE/OrderDate		List/Body/CustomerName	Company Ltd
ORDER/LINE/Adress	First Street 1		

Table 2.1: Example of problems that arise when trying to determine previous mapped paths from training data. Since the only possibility to create mappings is to identify data in the input file and find the same data in the output file, this method would generate some errors. First, it would assume that ORDER/LINE/qty would be connected to both and List/Body/quantity and List/Body/lineNmr, where the last mapping obviously is wrong. It would correctly construct the mapping ORDER/LINE/Id $\leftrightarrow$ List/Body/CustomerRef but since the formats for the date *20160125* changes to *160125*, the mapping between ORDER/LINE/DlvDate and List/Body/DeliveryDate would not be created, and the same principle would apply to the mapping between ORDER/LINE/Customer and List/Body/CustomerName. Even if a filter is used, where one manually removed obviously faulty mappings from training data and hence removed the mapping ORDER/LINE/qty $\leftrightarrow$ List/Body/lineNmr, this would be problematic in the scoring procedure. In the best case scenario, the algorithm would not suggest a mapping towards List/Body/lineNmr since that data is from an external source and should not be collected from the input file, but when comparing the original data file with the proposed one the validation script would notice that List/Body/lineNmr should have a data connected to it and hence give that suggestion an error even though it’s correct.

2.1.3 Mappings

A mapping is defined as a specification that a data value is moved from one specific path in one format, to another specific path in another format. In this project, the following three types of mappings is allowed:

- 1-0
- 1-n
- 0-1

A 1-0 mapping is when a path in the input format is not mapped to any path in the output format. A 1-n mapping is when one path in the input format maps to n different paths in the output format. For practical purposes, a 1-n map will be modeled as n number of 1-1 maps. The last case, 0-1 mappings, is when a path in the output format is not mapped and the output path is left empty.

In practice, more types of mapping can arise, for example one can split data from one path to two paths in the output format, or combine data from two paths in the input format to one single path in the output format. Data could also be appearing from other sources and put into the output format, for example a line number or a hash code. These cases will not be included in the scope of this project. The reasons for this are manifold, but mainly because it is hard to combine data values from different paths in a satisfactory way, and it would be a much bigger project if the algorithm should be able to know how to combine data, for example, an address, in a way that a human would do. Since this is a rare case it seemed like a reasonable limitation for the scope of this project. In some cases descriptions of the contents of a path is available at external sources. This information is not used in this project, mainly since these sources are not always available, and since this type of information sometimes requires a deeper understanding of the data integration procedure and the XML-schema.

The training data are tabulated such that each mapping between two paths is a row in a table, with the input and output paths and data values as columns.

Unmapped nodes are described by an empty output or input path. This means that each row could either contain a mapped pair of paths, where none of the paths is empty, or a row where one of the paths is empty. Since only mappings from a path with a connected data value is considered, each row with two non-empty paths will contain one data value (note that it is possible that paths that not maps still could contains a data value). The term mapping will be used when talking about these rows generated from a file pair. If nothing else is said, a mapping could be either an empty mapping, i.e. a mapping where one of the paths is empty, or a non-empty mapping, i.e. where neither of the paths is empty.

2.1.4 Set definitions

In the training data a set notation will be used to clarify which part of the data that is relevant in a specific case. The definition of each set can be seen in Table 2.2. The number of members of a set S is denoted $|S|$, otherwise standard set notation is used.

Denotation	**Explanation**
χ	A set of all XPaths
χ_i	A set of all XPaths in a file i
$\chi_i(w)$	A set of all XPaths in a file i containing the word w
M	A set of all mappings
$M(p_k)$	A set of all mappings containing the path k
$M(w_k)$	A set of all mappings containing a path with the word k
M_i	A set of all mappings in a file pair
$M_i \setminus \{0\}$	A set of all non-empty mappings in a file pair
F	A set of all file pairs
F_i	A specific file pair i
f	A set of all files
f_i	A specific file i

Table 2.2: A table of different set definitions used to explain what data that is used in the different steps

2.2 Statistical models

To decide if one should map two paths to each other or not, one needs to know which paths that has a connection to one another. The connection is in some sense based on a semantic connection between the paths, i.e. they describe the same data. It is not a semantic connection in the same way as one would associate a semantic connection in a natural language, but rather a semantic connection in the context of an XML-file. For example, a root node in one document could be completely different from a root node in another document, but in a way they are describing the same type of node in the XML-tree, and hence they are connected.
In the same way information regarding the individual words needs to be extracted, since even if each path has a meaning, each word in each path is important and can be used to decide if two paths has a similar meaning or not. This is done by computing the mapping probability between a path or a word, using the training data.

2.2.1 Conditional probability

To decide if two paths in an XML sheet should be mapped or not, it is necessary to know if they have been mapped earlier, and if so, how frequently. The interesting mapping probability in this case is the conditional mapping probability, that is, the probability that path p_n and path p_m is mapped to one another under the condition that both are present in the file pair to be mapped. The symmetric conditional mapping probability between two paths p_n and p_m in a file pair i is then defined according to Equation (2.1), which is the number of times they are mapped in the file pair, divided by the number of times they are mapped to anything in that file pair.
This probability is then averaged over all file pairs where the condition is fulfilled according to Equation (2.2), which is the final mapping probability used in the

model.

$$P^{(i)}(p_n \leftrightarrow p_m | p_n, p_m) = \frac{|M_i(p_n) \cap M_i(p_m)|}{|M_i(p_n) \setminus \{0\}| + |M_i(p_m) \setminus \{0\}| - |M_i(p_n) \cap M_i(p_m)|} \tag{2.1}$$

$$\bar{P}(p_n \leftrightarrow p_m | p_n, p_m) = P(p_n \leftrightarrow p_m | p_n, p_m) = \frac{1}{N} \sum_{i}^{N} P^{(i)}(p_n \leftrightarrow p_m | p_n, p_m) \tag{2.2}$$

In the same way, the conditional mapping probability between words are computed, with the difference that only the number of mappings where the paths is containing the words are counted, according to Equation (2.3), which are then averaged over all file pairs in the training set where the condition is fulfilled.

$$P^{(i)}(w_n \leftrightarrow w_m | w_n, w_m) = \frac{|M_i(w_n) \cap M_i(w_m)|}{|M_i(w_n) \setminus \{0\}| + |M_i(w_m) \setminus \{0\}| - |M_i(w_n) \cap M_i(w_m)|} \tag{2.3}$$

2.2.2 Information density

To distinguish between the members in a set of data, some parts of each individual set member will be more important than others. This is denoted as that the parts have different information density and should be weighted differently when determining the probability for mapping. In this case, the set is an output file or an input file, and the set members consists of the paths in the file. The parts of each member is the words in the paths.
The model for determining the weight of a certain word in a file is based on the inverse document frequency (IDF) measurement, commonly used in information retrieval, for weighing the relevance of a word [11].
The density of a word will be dependent of how often the word is present in the file. A word that is present in every path will be worthless when distinguish between paths, and hence get a density score of zero, but a word that only occurs once in a large file will be weighted much higher. The information density is computed according to Equation (2.4) and is computed individually for an output file or an input file, and does so only in the models and is hence not part of the training process or dependent of the training data.

$$S_i(w_j) = \log \frac{|\chi_i|}{|\chi_i(w_j)|} \tag{2.4}$$

2.2.3 Standard error of the mean

The standard error of the mean is denoted $SE_{\bar{x}}$ and defined according to

$$SE_{\bar{x}} = \frac{s}{\sqrt{n}}$$

where n is the sample size and s is the sample standard deviation defined as

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

2.3 Scoring

To be able to evaluate the models and draw conclusions about the performance, the suggested mapping given from the model is compared to the actual mapping. For each mapping of a file pair, each individual mapping of two paths is divided into different classes (see Table 2.3), depending on if the map is correct or not. In the case of the mapping being correct, that scenario is divided into two new classes, depending on if the map is a True Positive (TP) or a True Negative (TN). In the case of the mapping being incorrect, that scenario is divided into two other main classes, depending on if the map is a False Positive (FP) or a False Negative (FN).

Class	**Map type**	**Explanation**
Class 1	True Positive (TP)	A mapping $a \rightarrow b$ is done correctly
Class 2	False Positive (FP)	A mapping $a \rightarrow b$ is incorrectly done
Class 3	True Negative (TN)	A mapping $0 \rightarrow b$ is done correctly
Class 4	False Negative (FN)	A mapping $0 \rightarrow b$ is incorrectly done

Table 2.3: Class 1, 2, 3 and 4 are the main types of classification that the model can do. Class 2 has been divided into a and b so that one more clearly can see the types of errors that the algorithm does.

Note that the classification only concerns the mapping of the output paths, and not the input paths. The reason for this is that an input path can be mapped several times, but an output path can only be mapped at most once, and hence the number of output paths will always be the same in the suggested mapping and the correct mapping, but the number of input paths will not. Also, if an input path is mapped incorrectly it will result in an output path being mapped incorrectly, and if one would compare both input and output one would receive two errors for each faulty mapping.

2.3.1 F_β-score, precision and recall

Since it is a multiclass classification, and each classification is not considered equivalently good or bad, the F_β measure is used to decide how the models performs. The F_β measure is a way of weighting the importance of some classifications more than other. The F_β measurement is defined according to [8] as:

$$F_\beta = (1+\beta^2)\frac{\text{Precision} * \text{Recall}}{\beta^2 * \text{Precision} + \text{Recall}} \tag{2.5}$$

where we have

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.6}$$

and

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.7}$$

A False Positive is considered worse that a False Negative, for two main reasons:

1. A mapping that is done when it shouldn't, means that the model gets faulty data. A mapping that is not done when it should, means that the model does not get enough data. Since faulty data is considered a worse error in the model than incomplete data, a False Positive is considered worse than a False Negative.
2. In reality, the algorithm is supposed to make a guess of the mapping, so that an expert can save time in the mapping process. It is worse for the expert to have to validate a lot of mappings that should not have occurred than to be able to trust the performed mappings, and then add mappings that was not suggested.

The F_β-score is commonly used with one of three different values of β; 0.5, 1 or 2 in order to put more emphasis on precision, equal emphasis or more on recall. To put more emphasis on precision, and thereby weigh an error occurred from a False Positive higher than an error occurred by a False Negative, a value of $\beta = 0.5$ is used. If nothing else is mentioned, $\beta = 0.5$ will be used further wise.

2.4 Graph theory

The graph $G(V, E)$ is defined as a set of positive integer valued node indices V together with the set of edges E consisting of the connections between the nodes. The real valued edge between node $i \in V$ and $j \in V$ is denoted as $e(i, j) \in \mathbb{R}^+$. If nothing else is mentioned, a graph $G(V, E)$ will be assumed to be symmetric, i.e. $e(i, j) = e(j, i)$.

2.4.1 Shortest distance

The shortest distance problem between a node s and a node t in a graph can be formulated as a discrete linear programing problem as follows

$$x_{ij} \in \{0, 1\} \ \forall\, i, j \in V \tag{2.8}$$

$$\text{Minimize} \sum_{(i,j)\in E} x_{ij} w_{ij} \tag{2.9}$$

subject to

$$\sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & \text{If } i = s \\ -1, & \text{If } i = t \\ 0, & \text{otherwise} \end{cases} \quad \forall\, i \in V \tag{2.10}$$

where $w_{ij} = e(i, j)$ is the cost function for the edge between node i and j. In other words, the shortest distance problem is the task of finding the path between two nodes in such a way that the cost is as low as possible, were the cost is the sum of the edge weights on that path.

2.4.2 Maximum flow

The maximum flow problem can be stated according to [10] as follows:
An $s - t$ flow is a function that maps each edge e into a non-negative real number $f : E \to \mathbb{R}^+$, where the value $f(e)$ represents the amount of flow carried by edge e and $c(e)$ is the maximum capacity on edge e. A flow must satisfy the following conditions:

$$0 \leq f(e) \leq c(e)\ \forall e \in E \tag{2.11}$$

$$\sum_{\text{e into v}} f(e) = \sum_{\text{e out of v}} f(e)\ \forall v \in V \setminus \{s, t\} \tag{2.12}$$

and define $f^{out}(v) = \sum_{\text{e out of v}} f(e)$ and $f^{in}(v) = \sum_{\text{e into v}} f(e)$
The maximum flow problem is then, given a network graph, to ”arrange the traffic so as to make as efficient use as possible of the available capacity”.

2.5 n-grams

An n-gram is a contiguous sequence of n items from an ordered set (see Table 2.4). In our case the items are letters or other symbols in the data values. By collecting statistics of the occurrences of different n-grams into histograms new items can be compared to earlier ones by calculating the distance between their histograms.

n-grams of the word “example”		
1-grams	2-grams	3-grams
e	ex	exa
x	xa	xam
a	am	amp
..	..	..

Table 2.4: The table shows the first three n-grams in the word “example” for $n = \{1, 2, 3\}$.

2.5.1 Kullback–Leibler-divergence

The distance between two histograms can be calculated by the symmetrized version of the Kullback–Leibler (KL) divergence (2.13), $D_s = D(P, Q) + D(Q, P)$. [2] which is a measure of the relative information gain when approximating Q with P.

$$D(P,Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{2.13}$$

Where Q and P are the histograms of the n-grams. The order of these does not matter because of the symmetry $D_s(P,Q) = D_s(Q,P)$.
If some n-gram i is absent in one of the histograms its zero-count is replaced with a small value $\epsilon = 10^{-6}$ to make the logarithm well defined.

3 Methods

In this chapter the different methods used to create, train and validate the models that has been developed during this project will be presented.
It will presented how the training data is generated from the available sources, how errors in this process is handled and how the models are trained. It will also present the process of validating the models and how to decide the optimal parameters for the models.
Three independent models for determining the probability to map paths will be introduced, where the first is based on a minimum distance approach, the second is based on a maximum flow approach and the third is based on the n-gram distribution of data values connected to paths. The models has been created for three different main purposes. The first one is a simple model that can predict the mapping probability well if the paths has been seen before, but fails as soon as a path is a little bit different from any known path. The flow model is using the individual words instead of the complete paths, and can hence predict mappings between paths even if not all words in the path has been seen before. The third model is based solely on how the data values is mapped on previous paths, and is trying to map paths depending on if they previously has been mapped with similar types of data. This model is not thought of as doing any prediction alone in the mapping process, but to ensure that data values is not mapped to paths not usually containing that type of data. For example, an address should not be mapped to a path usually containing a phone number. The second purpose with this model, is to be able to insert new unseen paths and words into the other models. Since a word can be quite alike another word without having the same meaning, one could avoid connecting faulty words to the system if the data type do not match.

3.1 Generating training data

The raw training data consists of a large number of pairs of XML-files which has been previously mapped. To make this data usable the files are parsed into a table. Each path in the input format is identified and is considered mapped to a node in the output format if they are connected to the same data value. The resulting table lists the paths of the leaf nodes in the input and output files, their data value and an ID number describing in which file pair the mapping was found. Nodes that the process did not find a match for are considered unmapped, and is also listed in the table, but with one of the paths missing.
Some of the information stored in the XML-structure is lost in this parsing process,

such as element attributes, which is rare in the training set and does not provide that much information, and the tree structure, since occurrences of equally named nodes are not encoded in the specified data format.

3.1.1 Errors in the generated training data

This method is problematic when data values are not unique. If multiple nodes have the same value in the two files all of them are considered connected. It is essential that this can be managed, since if the models are trained with errors in the training data, chances are that these errors will make the models to do faulty mappings.
To manage these errors, faulty mappings are manually recorded. Since specific errors may arise multiple times, all these errors can be managed by just identifying one occurrence of that particular error. To find errors, all unique mappings from one path to another, ignoring data values, are inspected. Note that this is possible only since the number of unique path mappings is small. If data sets containing a larger variety of paths, this approach would not work in practice and another way of creating mappings for training would be needed.
Another problem with this approach of generating data is if the data value somehow is altered between the two files. In that case, two nodes that should be connected will not be so in the training data. This could for example happen if the two formats use different decimal marks or different date formats. This does not cause as much of a problem since the data will be missing instead of incorrect, as in the previous example..

3.1.2 Correcting faulty training data

By identify each unique mapping from the training data, one can manually remove mappings that is obviously faulty from the set. Doing so, the model will not be trained on faulty mappings. The validation procedure will not be affected by this procedure, since the validation data is generated in another way.

3.1.3 Generating validation data

To create validation data that is as close to the input the model would get in production, the validation data consists of all possible number of paths in the input format, the data values connected to each input path, and all possible number of paths in the output format. To be able to validate a mapping done by the model, this set is also containing all connected data values to the path in the output format. Note that the data values connected to the paths in the output format is never used by the model when determining the mapping, but only when validating if the mapping is correct or not.

3.1.4 Sampling data files

Most data sets contains thousands of previous mapped file pairs, and using them all in each iterations would not be possible, because of the computational time that would be needed. Instead random file pairs from the data is sampled for either

training or validation. The size of the sample is chosen in such a way that both the computational time for training and evaluation, and the estimated standard error is reasonably low. Since the algorithm computes the confidence level from each possible path in the input file towards each possible path in the output file, the computational time is approximately scaled as the square of the file sizes (if one approximates that each input and output file is about the same size, and that the size of a file is defined of the number of paths in the XML tree). Because of that, large files will take much longer to evaluate than the rest, which is why we for practical purposes only sample files smaller than some value N in the simulation processes.
In most cases it is found that a training set containing about 100 file pairs and a validation set containing 100 file pairs both fulfills these conditions. The maximum size allowed for files is usually set to 700 paths, and files larger than that is not used.

Data set	**Type of mapping**	**File pairs**	**Mappings**	**Correct mappings**[1]
data1_1	Input to intermediate	17 121	1 154 666	906 567
data1_2	Intermediate to output	10 276	194 538	118 380
data2_1	Input to intermediate	253	6624	4708
data2_2	Intermediate to output	11 622	375 645	287 141
data3_2	Intermediate to output	228	3425	2414

Table 3.1: The Table shows information regarding the five available datasets used in the project. The first column states the name that will be used to reference that set further wise, the second column gives information regarding if the set consists of mappings from a system to intermediate, or from intermediate to another system. The third and forth column presents the number of file pairs in the set, respectively the number of mappings (not unique) in the set. The last column states how many mappings that is left when the obviously faulty ones has been filtered out. In other words, it is not a guaranteed that every mapping in the set actually is correct.

3.2 Conditional mapping probabilities

To make the models learn which paths and words that has a relation to each other, the pre-mapped data is used to train the system. All unique mappings between paths in the training data is collected and the symmetric conditional mapping probability between each path is computed according to equation (2.1).
Since words are not explicitly connected in the same way as paths, one has to decide how to initialize this connection before it is possible to compute the conditional

[1]Correct mappings are the mappings that remain when the obviously faulty ones manually has been filtered out.

mapping probability between them. Two approaches for this has been tested with varying result. The first approach is to use the unique path mappings, and for each path mapping, each word in one path was connected to each word in the other path, and then the mapping probability was computed according to equation (2.3). It was found that this approach generated too many errors, and even though the mapping probability for the faulty mappings became low, the result was not satisfying. The final approach used was instead to create mappings of the words depending on their place in that path, starting from the last words in the path as seen in Figure 3.1. This still generates some uncertain mappings, but fewer than the previous approach. The mapping probability was then computed as before using Equation (2.3).
Note that these mappings not necessary is mapped according to semantic meaning in the classical sense, but rather an XML format semantics, where words used in the same way in the tree is considered equal. For example, the root element in the tree is mapped strongly to another root element, even if their semantic meaning in a natural language could be quite different. Note also that even if the construction of the word mapping are dependent of the position of the word in the path, the computed probability is not. All paths that contain a certain word will be included in the computation of the probability, independent of the position of the word.
A more detailed description of how the conditional probabilities between words and paths is computed can be seen in Algorithm 1 and Algorithm 2. The resulting mapping probability between each word is presented in Appendix **??**, where it in many cases is obvious that the words has a semantic connection, but in other cases it is not.

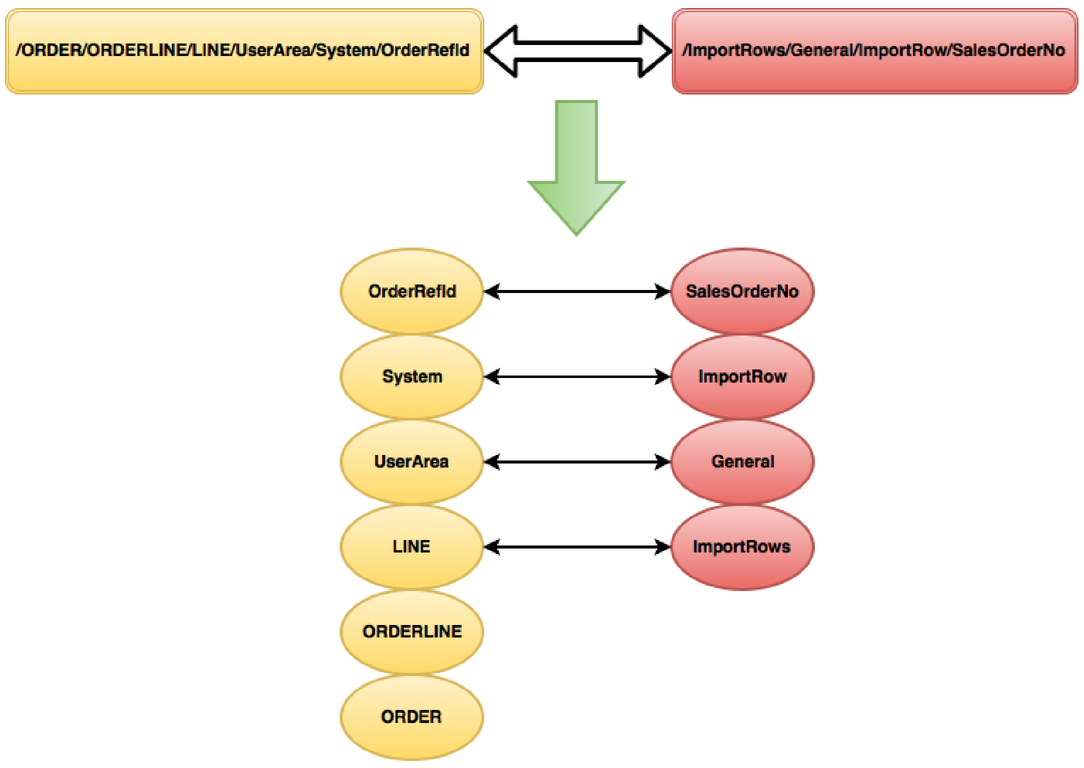

/ORDER/ORDERLINE/LINE/UserArea/System/OrderRefId
/ImportRows/General/ImportRow/SalesOrderNo
OrderRefId
SalesOrderNo
System
ImportRow
UserArea
General
LINE
ImportRows
ORDERLINE
ORDER

Figure 3.1: Process of converting a path mapping to several word mappings. Note that some words do not always get a mapping using this procedure.

Algorithm 1: ConditionalPathMapProbability

```
Data: File pair $M$ such that $M[i, \text{inPath}]$ and $M[i, \text{outPath}]$ is a correct,
      non-empty, mapping.
Data: $p1, p2$ such that $p1$ and $p2$ are paths.
Result: $P$ such that $p1$ and $p2$ is mapped with probability $P$ and that $P \geq 0$ if
        condition is fulfilled and -1 otherwise
begin
    $nP1 \longleftarrow 0$; /* Number of times $p1$ has been mapped */
    $nP2 \longleftarrow 0$; /* Number of times $p2$ has been mapped */
    $nP1P2 \longleftarrow 0$; /* Number of times $p1$ has been mapped with $p2$ */
    foreach row i in M do
        if $M[i, inPath] = p1$ or $M[i, outPath] = p1$ then
            $nP1 \longleftarrow nP1 + 1$
        if $M[i, inPath] = p2$ or $M[i, outPath] = p2$ then
            $nP2 \longleftarrow nP2 + 1$
        if $M[i, inPath] = p1$ and $M[i, outPath] = p2$ then
            $nP1P2 \longleftarrow nP1P2 + 1$
        else if $M[i, inPath] = p2$ and $M[i, outPath] = p1$ then
            $nP1P2 \longleftarrow nP1P2 + 1$
    if nP1=0 or nP2=0 then
        P = -1
    else
        P = $nP1P2/(nP1 + nP2 - nP1P2)$
    return P
```

3.3 Shortest distance model

The shortest distance model is based on the concept that if two objects never has been used to describe each other, but both object have been used to describe a third object, one would believe that the two objects have equal meaning and could be used to describe each other.
In this model, each unique path in the training data is modeled as a node in a graph with the inverse conditional mapping probabilities between the paths as edges. The idea is that clusters in the graph will be created in such a way that connected nodes have similar meaning.
This approach will be able to recognize paths that has been mapped before and map them again, but also have the possibility to map paths that has not been directly mapped to each other in the training data, but to some common path. It will not, however, be able to associate completely new paths with any path in the graph. If a path has switched the orders of two words, or if one of the words is a new one, even if that word is not of any importance for how it should be mapped (for example a root element), the model will not be able to associate that path with anything.
This will result in a model that will confidently map paths that has been seen before, but not at all when completely new data is entered.

Algorithm 2: ConditionalWordMapProbability

```
Data: File pair $M$ such that $M[i, \text{inPath}]$ and $M[i, \text{outPath}]$ is a correct,
      non-empty, mapping.
Data: $w1, w2$ such that $w1$ and $w2$ are words.
Result: $P$ such that $w1$ and $w2$ is mapped with probability $P$ and that $P \geq 0$ if
        condition is fulfilled and -1 otherwise
begin
    $nW1 \longleftarrow 0$; /* Number of times $w1$ has been mapped */
    $nW2 \longleftarrow 0$; /* Number of times $w2$ has been mapped */
    $nW1W2 \longleftarrow 0$;/* Number of times $w1$ has been mapped with $w2$ */
    foreach $row\ i\ in\ M$ do
        $path1 \longleftarrow M[i, inPath]$
        $path2 \longleftarrow M[i, outPath]$
        if $w1 \in path1$ or $w1 \in path2$ then
            $nW1 \longleftarrow nW1 + 1$
        if $w2 \in path1$ or $w2 \in path2$ then
            $nW2 \longleftarrow nW2 + 1$
        if $w1 \in path1$ and $w2 \in path2$ then
            $nW1W2 \longleftarrow nW1W2 + 1$
        else if $w1 \in path2$ and $w2 \in path1$ then
            $nW1W2 \longleftarrow nW1W2 + 1$
    if $nW1{=}0$ or $nW2{=}0$ then
        P = -1
    else
        P = $nW1W2/(nW1 + nW2 - nW1W2)$
    return P
```

3.3.1 Initialization of model

The model is created by using the computed mapping probability between known paths and initialize a graph with the edge weights as the inverse probability according to Equation (3.1), where $e(i, j)$ is the weight between path i and path j. For practical purpose, new unseen paths in the data that is to be mapped is inserted into the graph as unconnected nodes.

$$e(p_i, p_j) = [P(p_i \leftrightarrow p_j | p_i, p_j)]^{-1} \tag{3.1}$$

3.3.2 Computing confidence levels for mappings

The confidence level for mapping one path i in the input file to a path j in the output file, is computed as the inverse shortest distance between the two nodes (see Equation (3.2)), where $d(i, j)$ is the shortest distance between node i and node j in the graph, computed by Dijkstra's algorithm.

$$C(i, j) = [d(i, j)]^{-1} \tag{3.2}$$

In Figure 3.2 one can see an example of how a connected part of the distance graph can look. The arrows symbolizes that the paths has been mapped at least once in the training data, and the edge weight is the inverse of the computed mapping probability between them. For example one can see that the mapping probability between the path */PickingList/WMSPickingRoute/Customer* and */Order/ORDERHEAD/CustomerRef* is $\frac{1}{2}$.

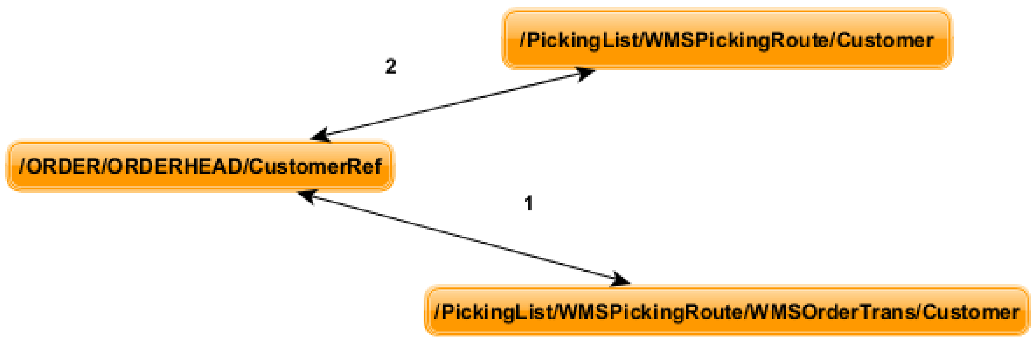

/PickingList/WMSPickingRoute/Customer
2
/ORDER/ORDERHEAD/CustomerRef
1
/PickingList/WMSPickingRoute/WMSOrderTrans/Customer

Figure 3.2: The Figure shows an example of the connections between paths in the distance graph. If one would compute the confidence level for mapping the path */PickingList/WMSPickingRoute/Customer* to */PickingList/WMSPickingRoute/WMSOrderTrans/Customer* one would get $[2 + 1]^{-1} = 3^{-1} = \frac{1}{3} \approx 0.33$.
Note that this graph has been made to illustrate an example, and the connections and edge weights shown in the graph is not real

3.4 Maximum flow model

In the shortest distance model, the concept behind the model was that information behind a mapping was the whole path, and that a path that looked similar, i.e. if only a few character was changed, it still was interpreted as a completely different path.
In the flow model, the idea is that information regarding the mapping is in the individual words, and that the individual words have connections towards other words with similar meaning. To compare two different paths, one would then look at how the words in one path has mapped towards words in another path, independently of the individual order. The idea is that using these mappings, the model would recognize paths that has the same words as earlier seen paths, even if the order would differ, or a path consisted of words from different paths, or the paths consisted of different number of words.

3.4.1 Initialization of model

The word mappings is extracted from the path mappings as seen in Figure 3.1, and the conditional mapping probability is computed according to Equation (2.3). One may note that it is somewhat problematic to determine the word mappings, since they do not exist naturally. The procedure of initialize the word mappings according to Figure 3.1 will result in connection between words that is not semantically equivalent. Although, since this will be more likely to happen to words early in the path, such as the root nodes, this will create a small cluster of words that might not have the same meaning, but they will neither be as important in the mapping procedure as the later words, which will have higher probability to be connected correctly. One should also be aware of that the connections between words that do not belong will be weaker, since they often also maps towards other words, and hence, they will not create errors in the model. Other ways of creating the mapping specifications between words has been to map each word in on path with each word in the other path. This does unfortunately create too many faulty connections and the final graph will not consist of clusters containing equivalent words, but rather of one large connected graph, which affects the results in a negative way.
Using the conditional probability between words as trained data, we create a graph system where the words act as nodes, and the edges between them is the conditional mapping probability, which act as a capacity in the graph. The hypothesis is that one could see the probability of mapping two paths, as a *probability flow* in the graph, and words with strong mappings has higher capacity of flow between them, than words that rarely maps to each other.
The trained graph is also consisting of a source node and a sink node, which is used as the paths one would compute the flow between. All words in the data set that is to be mapped, that is not already in the trained graph, will be inserted as unconnected nodes in the graph. The reasons for this is mainly two. The first is the practical one, that a word should be in the graph system when the flow is computed, instead of having to check if each word exists. The second is that if a word is not previously seen before, it can still contribute to the flow if the same word exists in the output

file. For example, if the word ”CUSTOMER” has not been seen in the training data, and hence do not have any connections in the graph, but in the files to be mapped, a path ending with ”CUSTOMER” exists in both files, a connection would be made between those path and hence contribute to a stronger flow.

3.4.2 Computing mapping probabilities

To compute a confidence level for mapping a path p_i to a path p_j, one starts with connecting one path as the source, and one path as the sink. The connections for the source and the sink is dependent on the words contained in the path. For the input path, the source will be connected to each word in the graph that is included in the input path, and for the sink, each word in the graph that is in the output path will get a connection.
Since all words in the path is not equally relevant for the mapping, all words have different possibilities to contribute to the flow. This is regulated by what capacities each connection from the source to the trained graph gets, and what capacity each connection from the trained graph to the sink gets. These capacities is set proportional to the information density of the individual words, computed according to Equation (2.4). The sum of the capacities from the source is then normalized to some factor small enough to not overflow the graph. The normalization is done since otherwise longer paths would in general get a higher flow than short paths (i.e. paths with fewer words in them).
The final confidence level is then computed as the maximum flow between the source and the sink, divided with the sum of the capacities from the source to the trained graph according to Equation (3.3), where $f(i, j)$ is the computed maximum flow between node i and j in the graph and $c(i, j)$ is the capacity between node i and node j.

$$C(p_i, p_j) = \frac{f(p_i = source, p_j = sink)}{\sum_k c(p_i = source, k)} \tag{3.3}$$

In Figure 3.3 one can see an illustration of how the paths is connected to the word graph in the flow model, and how the flow can run through the model. Note that in this example, the paths should probably be mapped, and the resulting confidence level will be high. In the case when the flow is computed between two paths that should not be mapped, the words will mostly be connected to different clusters in the graph, not connected to each other, and hence the maximum flow will be much lower than for this example.

3.5 Data value model

The data value connected to a node can also be used when determining how the mapping should be carried out. Different nodes usually have distinguishable types of data, for example a date, a contact name or some identification number. By classifying these data values, a mapping can be carried out without any knowledge of the structure of the input format. This method will not be able to distinguish for example a delivery date and an order date, or other similar data, but can still

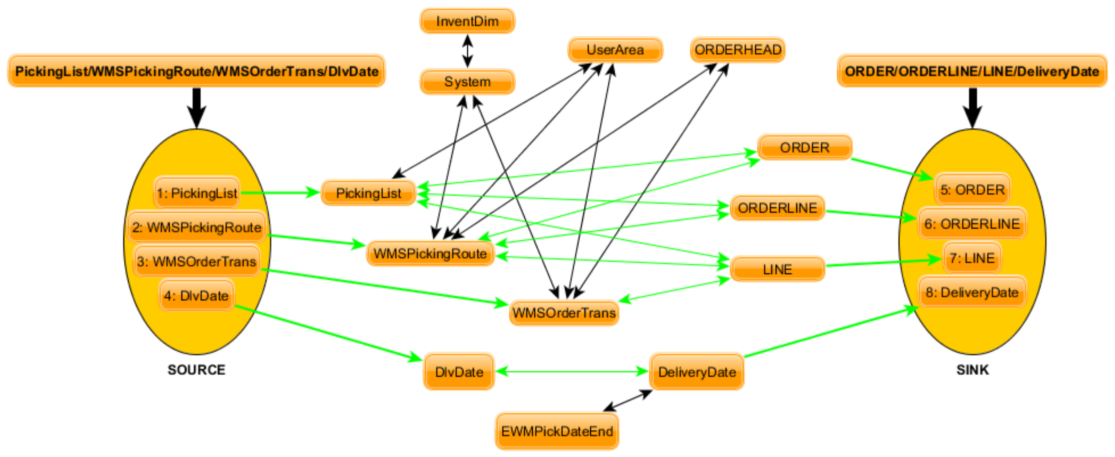


Figure 3.3: The figure shows an example of a confidence level computation for mapping the path *PickingList/WMSPickingRoute/WMSOrderTrans/DlvDate* to *ORDER/ORDERLINE/LINE/DeliveryDate*. One can see how the paths is set as a source respectively a sink, and the individual words is connected to the graph system. A maximum flow will then be computed between the source and the sink, which is illustrated by the green arrows. Note that the thickness of the arrows is not scaled depending on the conditional probability. The capacities between each edge will differ, especially between the source and the word graph, respectively the sink and the word graph, because of the difference in information densities for the words.

provide a guess and can be used in conjunction with the other models to improve the results.

3.5.1 Categorization using regular expressions

As a first step, data values are assigned to one of several format classes by using regular expression matching. This is mainly done to reduce computation times, as it takes some time to calculate the KL-divergences, but also to separate different types of data connected to the same nodes. These classes are listed in table 3.2

Class	**Description of string content**
Integer	Any number of occurrences of the digits 0-9
Decimal	Strings consisting of digits separated by one decimal point
Formatted number	Any of the digits as well as punctuation characters.
Alpha	Any characters. No spaces or other separators.
Alphanumerical	Any characters or digits.
Text	Characters and space-like separators.
Text-number	Characters, space and digits.
Generic	Anything not matched above.

Table 3.2: Format classes for data values, matched by regular expressions

3.5.2 n-grams

To get a confidence level for each map n-gram distances [4] from the data values to some predefined classes are calculated. The classes were defined from the unique last words in the intermediate format, assuming that a node always is connected to the same type of data. For each of these classes the L most common n-grams of lengths 1 to N where stored. The value of L was set to 1000, the default in the package used to implement the algorithm and N was set to 3, which proved to give satisfactory results.
When a new data value is inserted to the model, it is firstly assigned a format class, and then the distance D (2.13) to all classes that fits this format class are calculated. This distance is then converted to confidence levels by the activation function $C = e^{-\lambda D}$ where λ is a scaling factor. Figure 3.4 shows a simplified overview of this process.
This activation function was chosen because it maps the range of the distance function $[0, \infty]$ to the interval $[1, 0]$ and keeps the internal order of the results, in reversed order. λ can be arbitrary chosen but was set to $1/10$ to keep the results reasonably large. However, this choice will affect the optimal value of the threshold for the model.

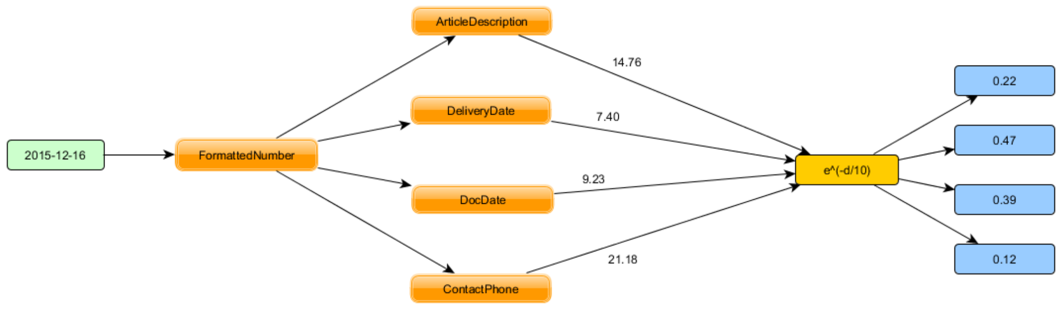


Figure 3.4: Overview of the data value model. A data value is entered, assigned to a format class and distances to classes associated with this format are calculated using KL-divergence. The numbers are then converted to a confidence level using the activation function. In this case, the date *2015-12-18* gets a confidence level of 0.47 to map to a *DeliveryDate* and 0.12 to map to *ContactPhone*

3.5.3 Connections between words and classes

A table of which words that have been mapped to the classes are created from the training set of mapped data. The classes are named from the last word in the paths they were created from and are connected to the last words from paths that they have been mapped to in the training set. A word can therefore have connections to multiple classes. For example, the word *id* would be connected to the classes *ContractId* and *OrderId*
The confidence level of mapping a data value from an input file to a given last word in the output file is taken as the maximum score of the classes the word in the output path is connected to.

3.6 Mapping using the models

The result from the models are the confidence levels for mapping each possible path from the input file to each possible path in the output file. The final decision of which paths that should be mapped to which takes this information into account, but also uses other factors, as decreasing confidence levels for already matched paths and thresholds to not include mappings with too low confidence level. A detailed description of the mapping procedure can be seen in Algorithm 3.

Algorithm 3: MapFilepair

```
Data: Model M containing mapping probability between each inPath and outPath
Data: Two parameters $\theta$ and $\eta$ such that $1 \geq \theta \geq 0$, $1 \geq \eta \geq 0$
Result: Matrix $M'$ such that each row in $M'$ contains the chosen mapping
begin
    $M' \longleftarrow \emptyset$
    while $M \notin \emptyset$ do
        Select mapping $m = \{inPath = p1, outPath = p2, p\}$ such that mapping
        probability $p$ is greater than, or equal to all other mapping probabilities.
        if $p \leq \theta$ then
            Exit loop
        Remove all mappings towards $p2$ from $M$.
        /* Decrease all probabilities from $p1$ */
        for $i$ = 1 to number of maps in M do
            if $M[i, inPath] = p1$ then
                $M[i, prob] \longleftarrow M[i, prob] * \eta$
        Add m to $M'$
    return $M'$
```

3.6.1 Decreasing confidence levels

The model allows at most one map towards each possible output path, but it allows each possible input path to be mapped to several different output paths. Since it could be the case of, for example, three identical input paths that should be mapped to three different output paths, we have to decrease the confidence level for all mapping that include a specific path each time that path is mapped, otherwise, one of the three paths could be decided to map to each and one of the three outputs, which would make multiple instances of data to be faulty mapped, and the remaining data to not be mapped at all.
To comprehend for this case, we update the confidence level from path i to path j each time we decide to map i to some path, according to Equation (3.4).

$$P(i \to j) \to \eta P(i \to j) \ \ \forall j \tag{3.4}$$

where $0 \leq \eta \leq 1$ is the parameter controlling how fast the confidence level decreases.

3.6.2 Mapping threshold

It is not necessary to map each path in the input format that contains data, to a path in the output format. It is sometimes the case that the data value is not supposed to be transferred, for different reasons. Because of this we will not allow a mapping to happen if the confidence level is lower than some threshold θ, $1 \geq \theta \geq 0$. Since the models works in different ways, the confidence levels are differently scaled and hence it is not certain that the same value of θ can be used in all models.

3.7 Validation process

The mapping suggestion returned by the algorithm is validated by comparing it to the original output file. The different possible cases are summarized in Table 2.3. The F-score (2.5) for each file pair is calculated, and then averaged over all files in the sampled validation set.
This validation process is problematic mainly because of the same reasons the process of generating training data is problematic; the correct mappings are not known.
What is known is how the output file is supposed to look, and it is possible to compare the result to this. The mapping algorithm has made a suggestion of were the data values from the inputs should end up, and by comparing the suggested places for the data values from the mapping algorithm with the actual output file, one can in most cases correctly classify a mapping.
One may note that there are still some problems with this procedure that can make a faulty classification in some cases. The four most common cases for when this could happen, together with the potential error classification, is listed in Table 3.3. Except for the potential errors listen in Table 3.3 it is possible that two identical paths in the input format can be proposed to be mapped to two identical paths in the output format. Since there are two different ways to perform this mapping (since the paths can be connected to different data), if the algorithm has proposed another mapping order than is in the output file, both mappings will be considered faulty. One should here note that it is not necessary that they should be considered faulty, since one could consider a case where the file consists of two groups of data describing two products that has been ordered, for example. The groups could contain information regarding the price, the quantity and the item number. Both these groups of information should be moved to the other format, and the most important thing is that the data groups ends up together. If one group ends up above the other, or the other way around is not of importance. As an example, in an invoice, it does not matter in which order the items are listed, as long as the correct price is connected to the correct item. The mapping algorithm could therefore suggest a mapping of another order than what the previous expert did, and therefore the mapping will be classified as an error even if it not necessary is incorrect. The complete procedure of determining the classification of a mapping can be seen in Algorithm 4.

Type of mapping	**Possible outcome**	**Classification**
The algorithm proposes faulty mapping with data value	If the data value for the correct mapping happens to be equal, the mapping will be faulty classified	Classified as Class 1 instead of Class 2
The algorithm proposes faulty mapping without data value	If the data value at the proposed endpoint has changed its format, and is not found in the input file, this will be faulty classified	Classified as Class 3 instead of Class 4
The algorithm proposes correct mapping with data	If the data value has changed its format, the classification will be faulty	Classified as Class 2 instead of Class 1
The algorithm proposes correct mapping without data value	If the endpoint possesses a data value that origins from other sources than the input file, but is identical to another value in the input file, the classification will be faulty	Classified as Class 4 instead of Class 3

Table 3.3: The Table lists the most common errors that can occur when trying to classify a proposed mapping.

Algorithm 4: ValidateMapping

```
Data: Table M containing proposed mappings
Data: Table C containing the original outputs
Data: Table I containing original inputs
Result: Table $M'$ containing the proposed mappings together with the
        classification of each mapping
begin
    $M' \longleftarrow \emptyset$
    foreach map in M do
        if Mapped data value equal to original data & data value non-empty then
            add map to M', set class 1
        else
            if A mapping is proposed then
                if Data value present in I then
                    add map to M', set class 2
            else
                if Data value present in I then
                    add map to M', set class 4
                else
                    add map to M', set class 3
    return $M'$
```

3.8 Implementation of the models

All programs are written in the R programming language, which is widely used for statistics. The language have an active community and there are a lot of third-party libraries (at the time of writing 7725 packages) available via Comprehensive R Archive Network (CRAN) which makes it easy to implement and test different models.
The main libraries used in this project are listed in table 3.4

Library	**Used for**
`sna` [3]	Maximum flow calculations
`igraph` [5]	Shortest distance calculations
`textcat` [9]	n-grams and KL-divergence

Table 3.4: Third-party R libraries used in models.

The basic structure of the test program is divided into five parts:
1. Sample training and validation data
2. Train models with training data
3. Calculating all pairwise confidence levels between all input and output nodes using the models and validation data
4. Propose a mapping using the confidence levels

5. Validate the mapping

When using the program in production only step 3 and 4 needs to be executed.

4

Experiments

In this chapter we will present the simulations that have been done to evaluate the models and determine how well they perform during different conditions. Most evaluation procedures will be parameter sweeps, since the mapping algorithm are using two different parameters, which optimal values are unknown. Little focus will be on the exact parameter values, since these are not going to be important when an expert is doing the mapping with suggestions from the models instead of the constructed mapping algorithm. What will be focused on is the highest achieved F_β-score, and how the class distribution looks like.
The models will be evaluated mainly in four different training environments. First, the training will be done on each of the datasets individually, and the validation data will be sampled from the same set that the training was sampled from. Second, the training will be performed from all available sets and the validation will be done on one set at a time. Third, the training will be performed on all datasets and the validation will be done on all datasets. The last training environment will be on all datasets but one, where the last dataset will be used for validation. These cases will together help to show how the models perform during optimal conditions, if the different sets interfere negatively with each other, and if the models are able to perform on completely new data.

4.1 Experiment setup

The setup for the experiments has been designed in such a way that the simulation environment should be as close to the real production environment as possible, and that information that wouldn't be available in reality is not available in the simulation.
All experiments are carried out for one model at a time, and each result is presented individually for each model. No experiment has been done with a combination of the models, since it has been assumed that the individual performance is more interesting, and a meta model, a combination of all models, would in many cases not make a difference in the current situation since the models have been designed to work in different situations. For example, a meta model would not just be a linear combination of the outputs from the various models, but rather use the models depending on the situation. For example, if the paths in the input file are known, and the paths in the output file are known, the distance model would be better to use to give suggestions of the mapping. In the case where new data is present, the flow model would probably be a better choice, since the path model would not be

able to associate them. In some cases it is possible that some data is known, and some is not, which gives a harder case, but it would both be possible to use a linear combination of the outputs, or to give all suggestions to the mapping expert. The data value model is not thought of to give suggestions on its own, since the error rate is too high. That model would instead be used if one would insert new words into the data that has not been seen before. Using a combination of string comparing algorithms and the data model, would prevent connecting the new word to a word in the graph that looks a bit alike, but maps completely different types of data. One example of this case could be if the word *qtyUOM* would be compared to QTY, which probably would get a good match in many string comparing algorithms, but since *qtyUOM* stands for ”Quantity Unit Of Measure” it would more likely contain data of the type *pcs*, while QTY rather would contain integer numbers, and thereby that connection could be avoided.

4.1.1 Experiment data

The data used in these experiments are divided into three parts, training, mapping and validation. The training sets are the data generated from identifying unique data in the file pairs and creating a mapping between each path that shares the same data value. This set is then filtered through a manually created filter containing all identified faulty maps. The faulty maps is generated since the assumption that a data value is unique is a bad assumption, since some values, mostly small integers, often are contained by multiple paths, not connected to each other.
The training data is sampled from five different available areas (see Table 3.1), which are assumed to have little in common with each other. Since most sets are too large to be used as a whole, parts of the set are sampled randomly to be able to receive a reasonably computational time. The sample size used is set in such a way that both a short enough computational time is received, and such that the standard error of the mean score is reasonably low.
The data set used for mapping using the model consists of all available paths in an input file with connected data values, and each possible path in the corresponding output file. The mission of the algorithm is then to determine which and how paths in the input format should be connected to the available paths in the output format. To validate the mapping, one can then compare where the algorithm put the data values and where the data values should have been.

4.1.2 Mapping process

The mapping process will be as equivalent to a production environment as possible, since the input is more or less the same as will be available in a real situation. The algorithm gets the possible paths in the input format with the connected data values, together with the possible paths in the output format. The mapping algorithm is then using the mapping probability computed by the models to determine which paths that should be mapped, and which paths that should be left unmapped (see Algorithm 3).

4.2 Parameter sweeps

The focus of this report has been to develop a system that can give an expert a guess for a mapping, it is not supposed to be able to create a mapping completely on its own. To be able to evaluate the models, it has been necessary to create a simple mapping algorithm that creates a complete mapping which can be compared with the correct mapping, using the F_β-score (2.5). This algorithm is foremost dependent of two parameters, a threshold θ to avoid mapping paths with too low confidence levels, and a parameter η that is used to reduce the probability for mapping the same path twice. To find the best possible score that can be reached using this algorithm, parameter sweeps are done. Since these parameter will not be as important in production, the main focus will be on the score received from the different models, and not as much on the actual parameter values used to receive that score. The models are trained using the available data and been ran and evaluated independently from each other. The training process differs to be able to draw conclusions regarding if the models benefit from being trained on small parts of data closely connected to the area it is supposed to work on, and if negative interference reveals when training on more data. The models are also evaluated on datasets they have never been trained on.

4.2.1 Sweeps of parameters θ and η

Two main parameters are used when deciding how the mapping from the models should look like, using the confidence levels computed from the different models. The first introduced parameter is the threshold θ, which decides how small the confidence level can be before it is decided not to map two paths. The other parameter is η, which is used to decrease the probability for mapping one particular path twice (see Equation (3.4)). Since these two parameters are tightly connected, sweeps have been done on the intervals $1 \geq \theta,\ \eta \geq 0$ with a step length of 0.1.
Note that these parameters are only important for the mapping algorithm, and the sweeps are done to find the optimal F_β-score, and not to find optimal values for the parameters, since these are not interesting when the mapping is handled by an expert.

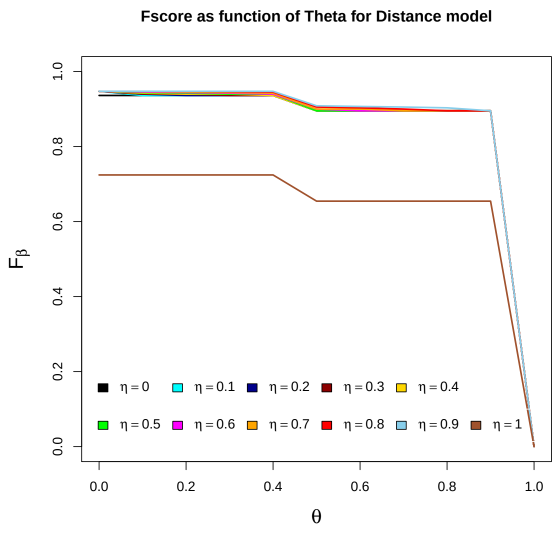
4.2.2 Training and evaluation on isolated datasets

In Figure 4.1, 4.2, 4.3, 4.4 and 4.5 one can see the F_β-score for the datasets `data1_1`, `data1_2`, `data2_1`, `data2_2` and `data3_2` respectively. 100 random file pairs were sampled from the particular set and used for training, and equally many were sampled and used for validation. As one can see, the F_β-score is higher and more stable for the distance model than the flow model, and the data value model performs worst, as expected. One can also see that it is not obvious at a first sight which parameter values that generates the highest F-score, but $\eta = 1$ generates the worst score in most cases. This is because the models always returns the same confidence level for identical inputs and outputs. In most cases like this, one would not want to map one input path to two identical outputs, but rather two identical inputs to two identical outputs. Without decreasing the probability the mapping algorithm will map the

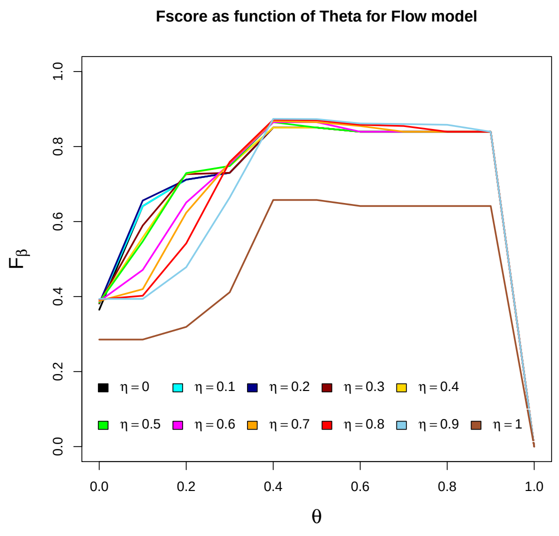
first input to all identical outputs. One can see that the optimal F_β-score is received for quite different parameters for the three models. The distance model gets the highest score for small values of θ, sometimes more or less independently of η, while the flow model more often receives highest score for larger values of θ. The main reason for this is that when the distance model gets any non-zero confidence level for mapping two paths that means that they have been mapped before, in some way, and probably should be mapped again. Meanwhile, the flow model can get a non-zero confidence level between paths that are probably not a match, since it finds connections at root level, and hence too small confidence levels must be discarded to receive a high score. The score of the data value model is almost independent of η. This is because the inputs are often unique makes the model return unique confidence levels.
In Table 4.1 one can see a summary of what values of θ and η that generates the highest F-score in these simulations. As one may notice, the optimal values differ from set to set. The hypothesis why this is that in some datasets the same path should map more than once with higher frequency than in other datasets. It is also a possibility that some graph systems for the words in the flow model are less connected when training on some sets than on others, which could explain why the θ parameter differs.

Model	**Dataset**	θ	η	F_β**-score**
Distance	data1_1	0	$0.9 \geq \eta \geq 0.1$	0.947
	data1_1	$0.4 \geq \theta \geq 0$	0.9	0.947
	data1_2	0	$0.9 \geq \eta \geq 0.1$	0.825
	data2_1	$0.2 \geq \theta \geq \eta$	$0.2 \geq \eta \geq 0$	0.885
	data2_2	0.4	0.9	0.817
	data3_2	0.1	0.3	0.671
Flow	data1_1	0.4	0.9	0.874
	data1_2	0.4	0.9	0.805
	data2_1	0.6	$0.6 \geq \eta \geq 0$	0.873
	data2_2	0.6	0.9	0.740
	data3_2	0.6	0.9	0.626
Data Value	data1_1	0.1	0.8	0.528
	data1_2	0.1	0.8	0.580
	data2_1	0.1	0.5	0.640
	data2_2	0.2	0.6	0.361
	data3_2	0.7	0.9	0.551

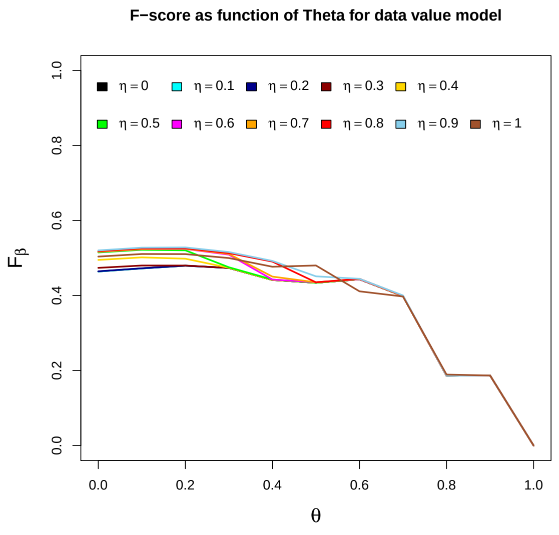
Table 4.1: The Table shows for which values of θ and η the optimal F_β-score is received, for $\beta = 0.5$. The data is computed using 100 random sampled files for training and 100 random files for validation.



(a) Parameter sweep for distance model

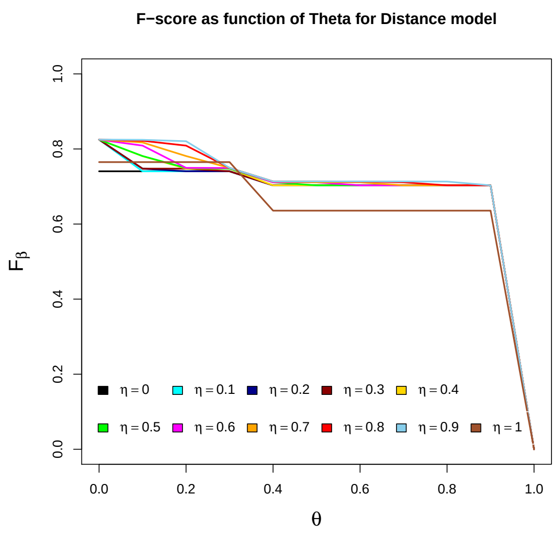


(b) Parameter sweep for flow model

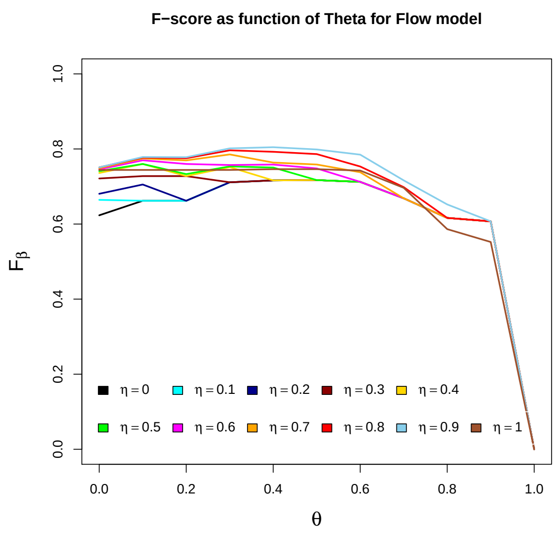


(c) Parameter sweep for data value model

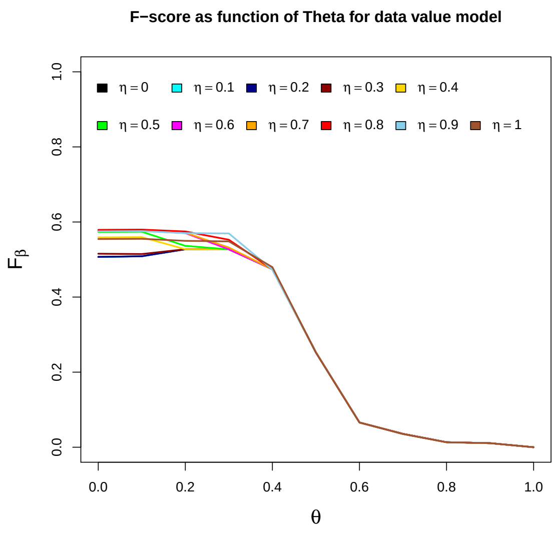
Figure 4.1: The figure shows how the F_β-score changes when the parameters θ and η changes. The simulation is done on data set `data1_1` and is trained on 100 random files and validated on 100 other files.



(a) Parameter sweep for distance model

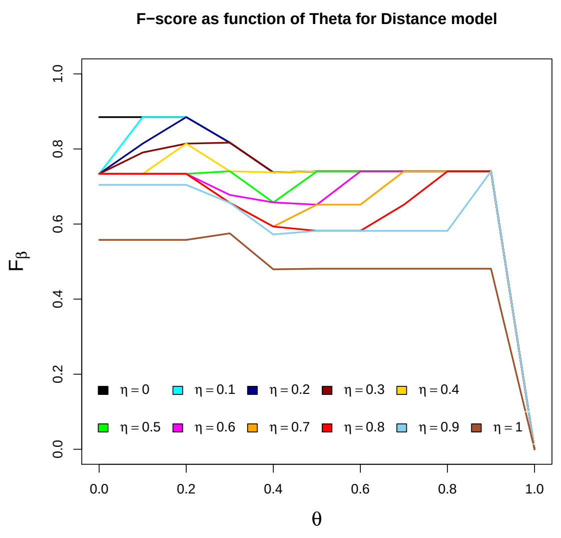


(b) Parameter sweep for flow model

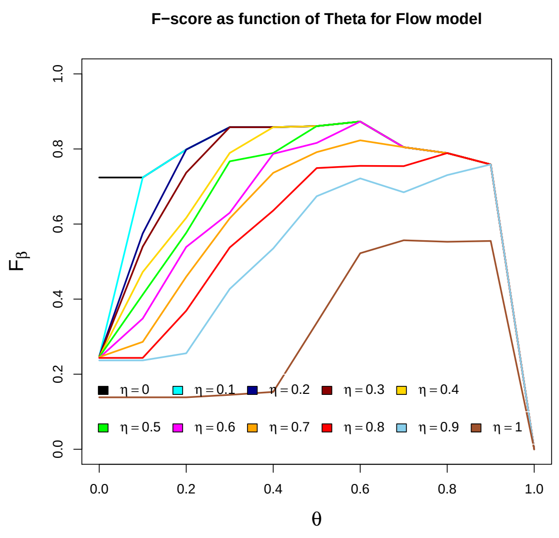


(c) Parameter sweep for data value model

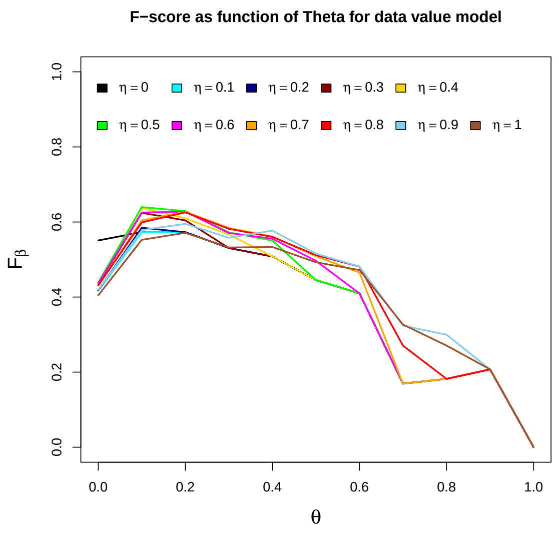
Figure 4.2: The figure shows how the F_β-score changes when the parameters θ and η changes. The simulation is done on data set `data1_2` and is trained on 100 random files and validated on 100 other files.



(a) Parameter sweep for distance model

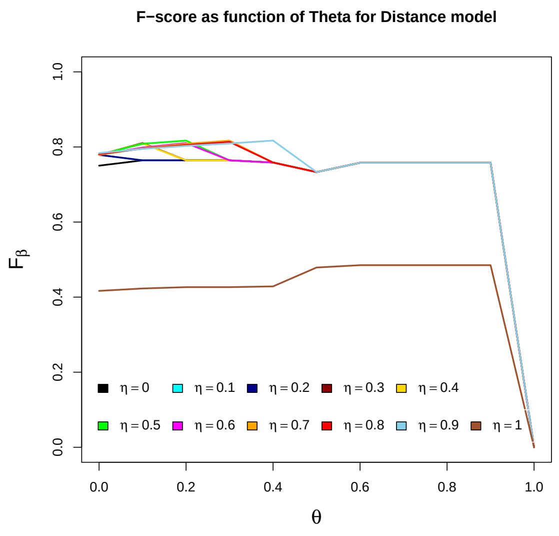


(b) Parameter sweep for flow model

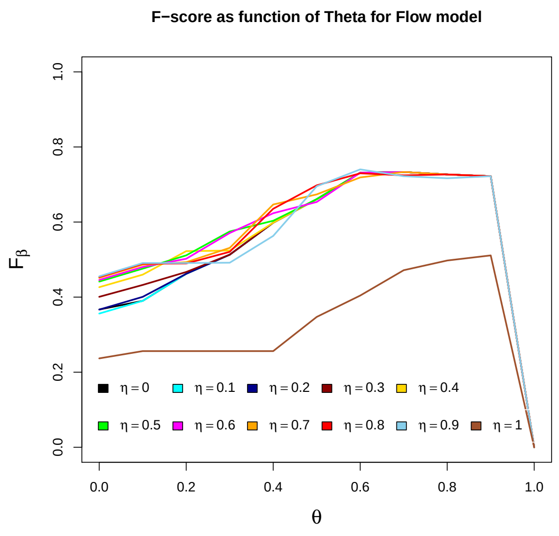


(c) Parameter sweep for data value model

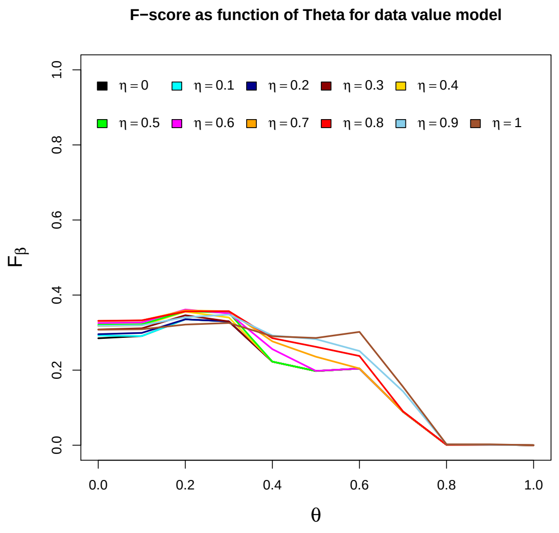
Figure 4.3: The figure shows how the F_β-score changes when the parameters θ and η changes. The simulation is done on data set `data2_1` and is trained on 100 random files and validated on 100 other files.



(a) Parameter sweep for distance model

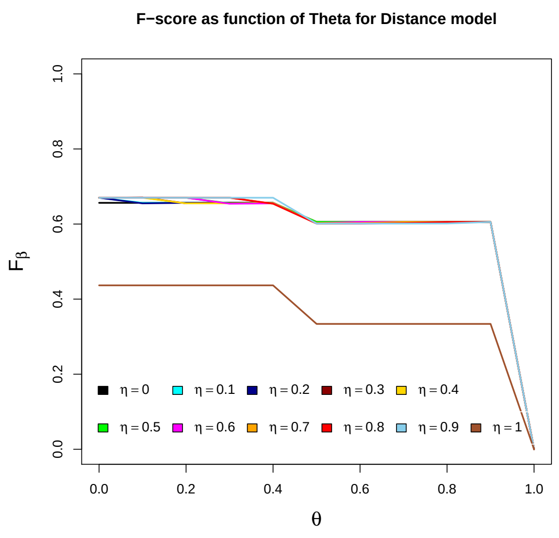


(b) Parameter sweep for flow model

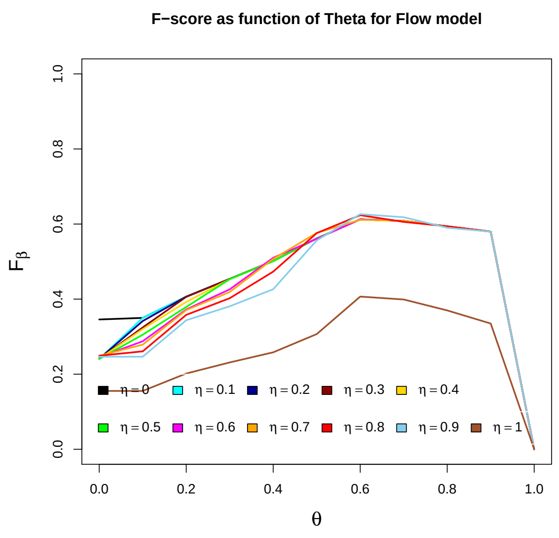


(c) Parameter sweep for data value model

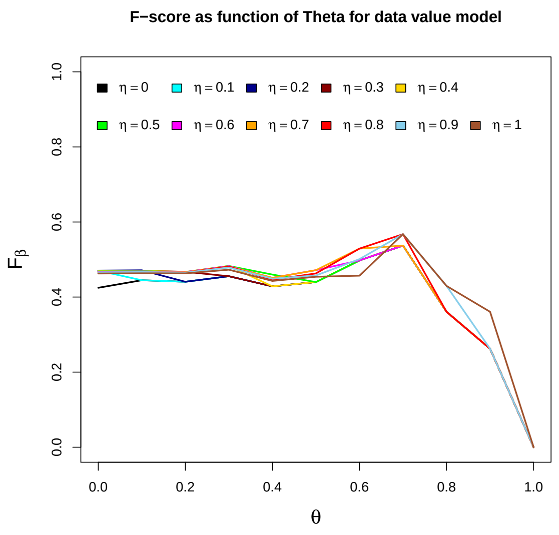
Figure 4.4: The figure shows how the F_β-score changes when the parameters θ and η changes. The simulation is done on dataset `data2_2` and is trained on 100 random files and validated on 100 other files from the same set.



(a) Parameter sweep for distance model



(b) Parameter sweep for flow model



(c) Parameter sweep for data value model

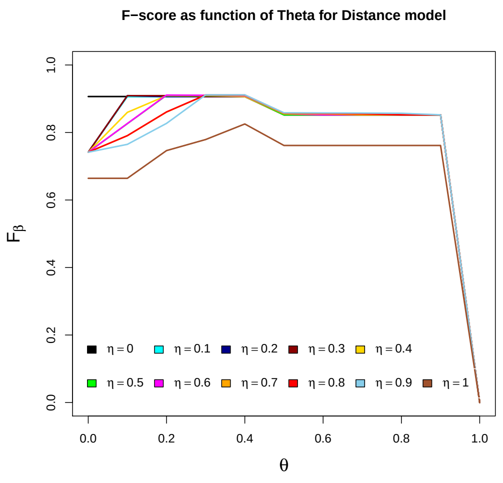
Figure 4.5: The figure shows how the F_β-score changes when the parameters θ and η changes. The simulation is done on data set `data3_2` and is trained on 100 random files and validated on 100 other files.

4.2.3 Finding negative interference between datasets

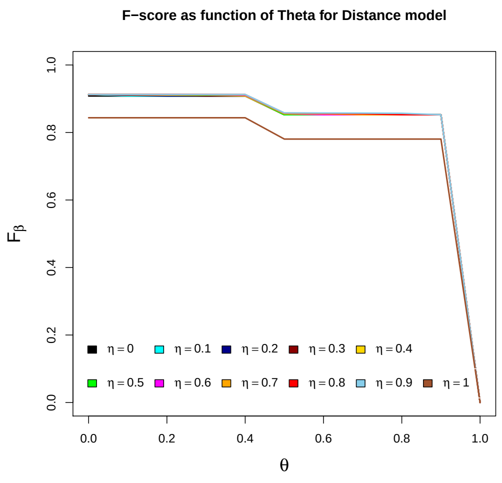
To examine if the different datasets contribute to mapping probabilities that negatively interfere with each other, the models are trained and validated for two different cases. The first case is the standard case when 100 random file pairs are sampled from one dataset and used for training, and 100 file pairs are randomly sampled from the same dataset and used for validation. The second case is when 100 file pairs are randomly sampled from each if the available datasets, and 100 file pairs are sampled from one dataset and used for validation. The resulting sweeps can be seen in Figure 4.6, 4.7, 4.8, 4.9 and 4.10. In Table 4.2 one can see the highest recorded F_β-score for the different models, together with the computed standard error of the mean.

		Train on all, validate on one			Train on one, validate on one		
Model	Dataset	$\bar{F}_{0.5}$-score	s	$SE_{\bar{F}}$	$\bar{F}_{0.5}$-score	s	$SE_{\bar{F}}$
Distance	Data1_1	0.911	0.019	0.002	0.913	0.021	0.002
	Data1_2	0.863	0.108	0.011	0.859	0.111	0.011
	Data2_1	0.670	0.055	0.006	0.795	0.031	0.003
	Data2_2	0.657	0.088	0.009	0.777	0.122	0.012
	Data3_2	0.859	0.137	0.014	0.852	0.138	0.014
Flow	Data1_1	0.755	0.036	0.004	0.781	0.030	0.003
	Data1_2	0.859	0.118	0.012	0.855	0.118	0.012
	Data2_1	0.564	0.034	0.003	0.742	0.030	0.003
	Data2_2	0.582	0.107	0.011	0.630	0.112	0.011
	Data3_2	0.712	0.109	0.011	0.732	0.124	0.012
Data value	Data1_1	0.355	0.070	0.007	0.565	0.081	0.008
	Data1_2	0.580	0.162	0.016	0.557	0.141	0.014
	Data2_1	0.456	0.081	0.008	0.643	0.079	0.008
	Data2_2	0.229	0.112	0.011	0.341	0.162	0.016
	Data3_2	0.487	0.188	0.019	0.585	0.080	0.008

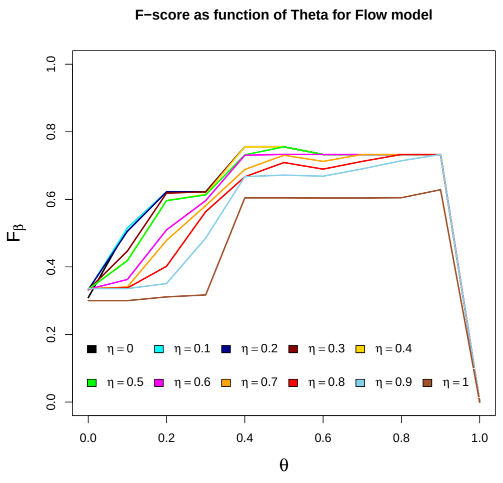
Table 4.2: Highest F_β-score received during parameter sweep for all datasets. The presented values comes from two different simulations. In the first one, the training process was carried out on each available training set, that is, 100 file pairs was randomly sampled from each dataset and then 100 random files was sampled from the specified dataset to be used for validation. In the second simulation, the training was carried out on 100 randomly sampled file pairs from the specified dataset, and another 100 file pairs from the same dataset was randomly sampled to be used for validation. One can see that in some cases, it differs very little depending on if all datasets was used or only the dataset used for validation, but in the cases when it differs more, the highest score has been received when all training data is from the same set used for validation. The conclusion that can be drawn is that it is a possibility that negative inference occurs when training on data from different areas.



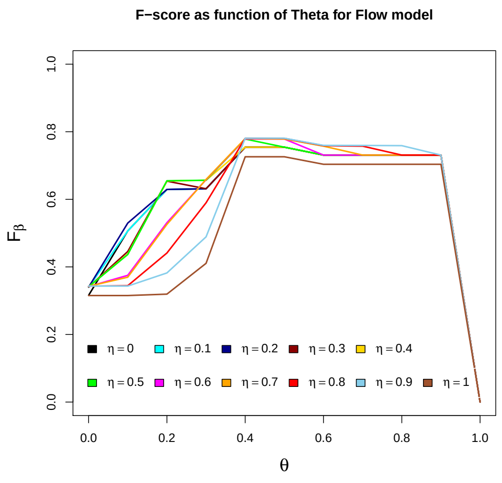
(a) Distance model when training on all datasets and validating on `data1_1`



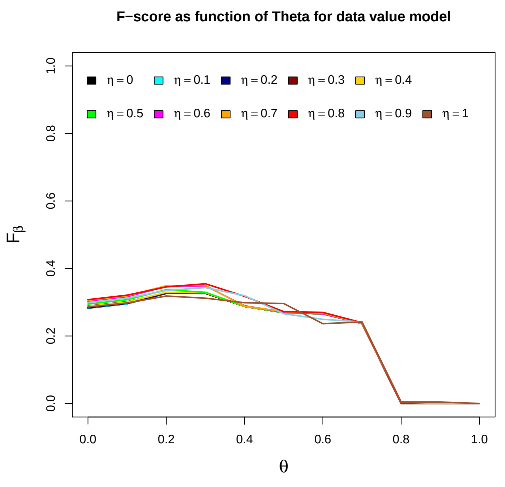
(b) Distance model when training on `data1_1` and validating on `data1_1`



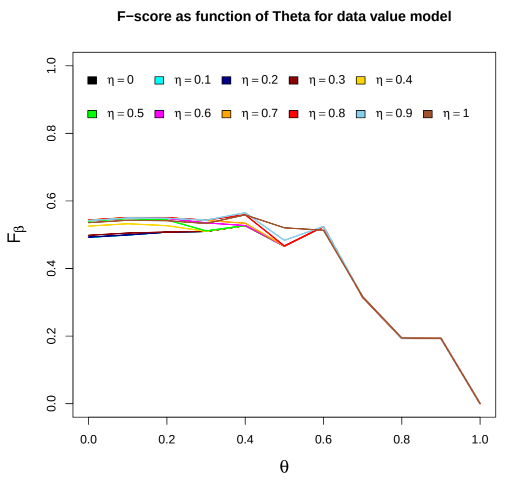
(c) Flow model when training on all datasets and validating on `data1_1`



(d) Flow model when training on `data1_1` and validating on `data1_1`

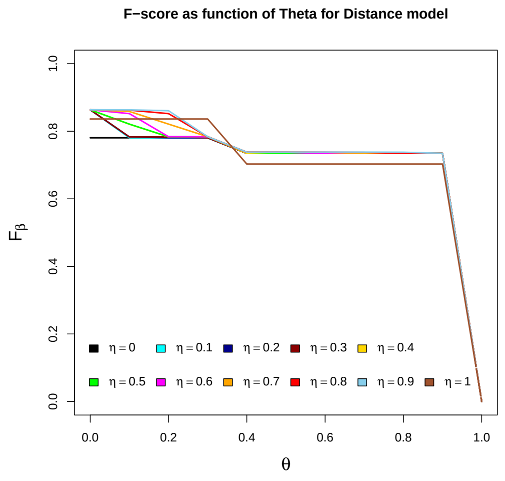


(e) Data value model when training on all datasets and validating on `data1_1`

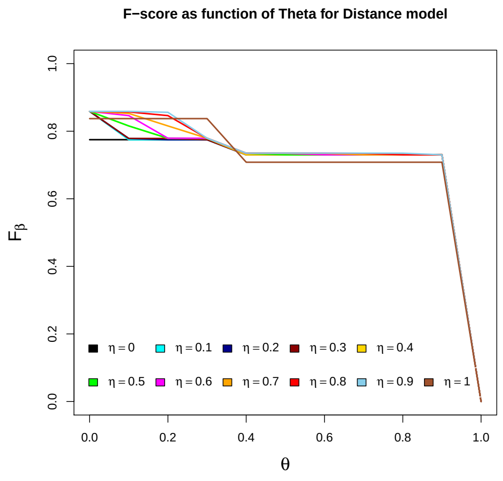


(f) Data value model when training on `data1_1` and validating on `data1_1`

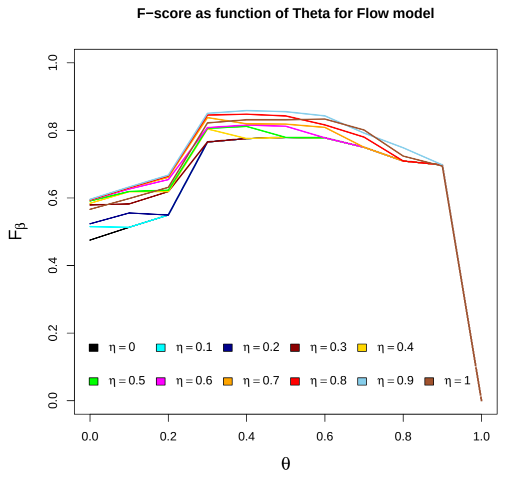
Figure 4.6: The Figure shows how the F_β-score changes depending on the parameters θ and η for 4.6(a) Distance model when all datasets is used for training, 4.6(b) Distance model when only `data1_1` is used for training, 4.6(c) Flow model when all datasets is used for training, 4.6(d) Flow model when only `data1_1` is used for training, 4.6(e) data value model when all datasets is used for training, and 4.6(f) when only `data1_1` is used for training. When training on all datasets, 100 random file pairs was sampled from each set. In the case of training only on `data1_1` one hundred file pairs was randomly sampled. In both cases 100 file pairs was randomly sampled for validation from `data1_1`. The validation was sampled from the remaining file pairs in the set not used for training.



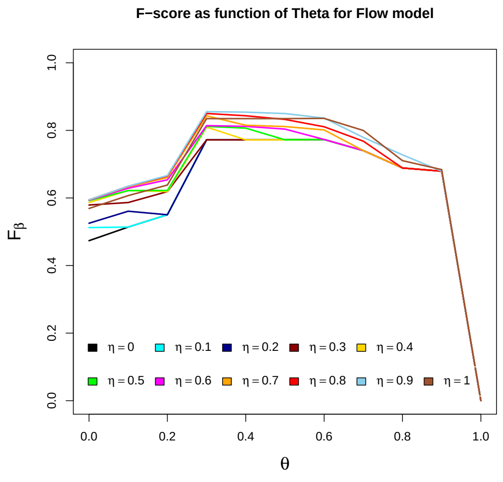
(a) Distance model when training on all datasets and validating on `data1_2`



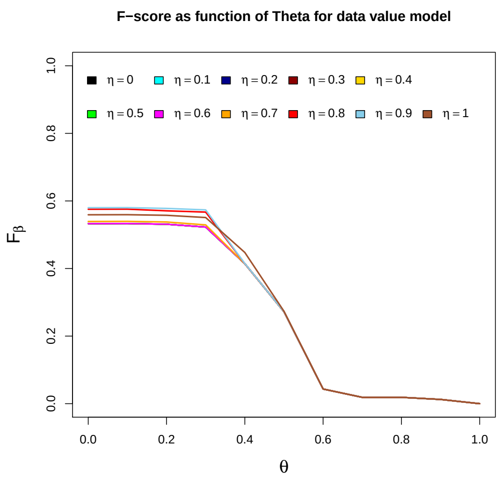
(b) Distance model when training on `data1_2` and validating on `data1_2`



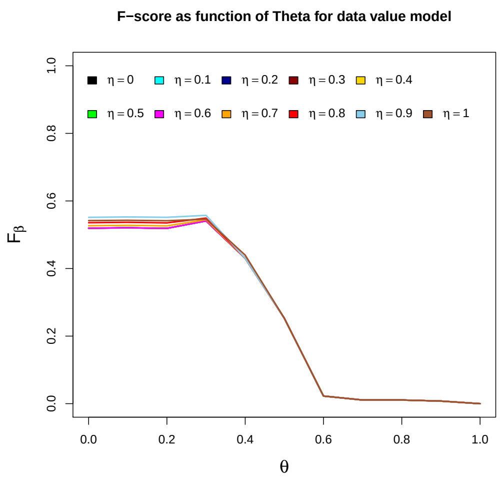
(c) Flow model when training on all datasets and validating on `data1_2`



(d) Flow model when training on `data1_2` and validating on `data1_2`

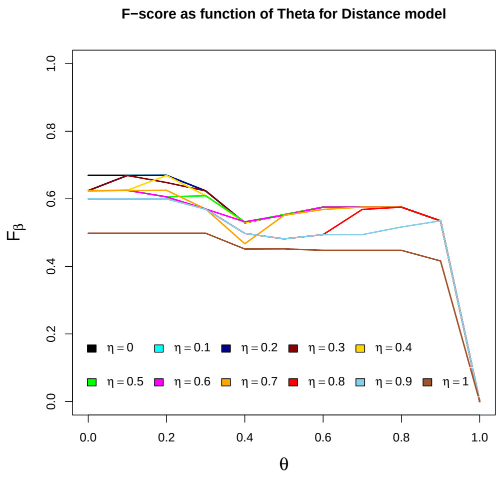


(e) Data value model when training on all datasets and validating on `data1_2`

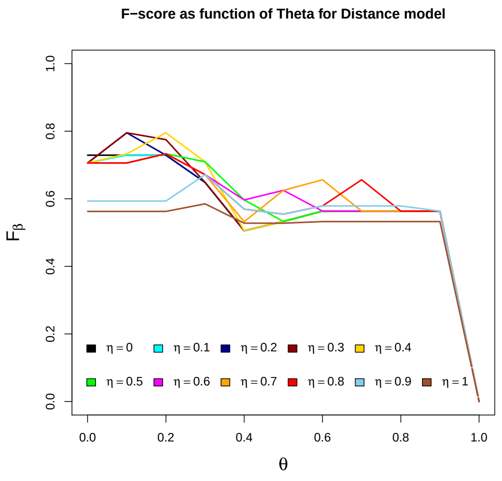


(f) Data value model when training on `data1_1` and validating on `data1_2`

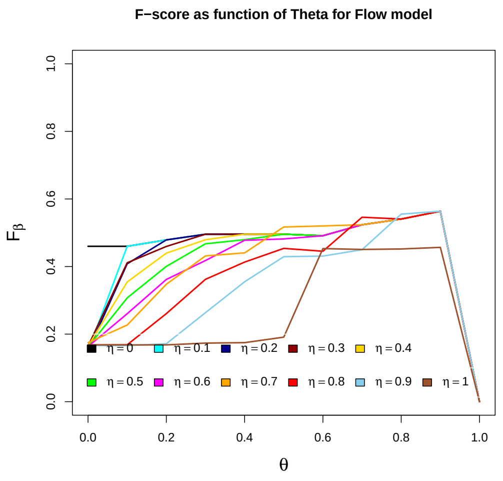
Figure 4.7: The Figure shows how the F_β-score changes depending on the parameters θ and η for 4.7(a) Distance model when all datasets is used for training, 4.7(b) Distance model when only `data1_2` is used for training, 4.7(c) Flow model when all datasets is used for training, 4.7(d) Flow model when only `data1_2` is used for training, and 4.7(e) data value model when all datasets is used for training, and 4.7(f) when only `data1_2` is used for training. When training on all datasets, 100 random file pairs was sampled from each set. In the case of training only on `data1_2`, one hundred file pairs was randomly sampled. In both cases 100 file pairs was randomly sampled for validation from `data1_2`. The validation was sampled from the remaining file pairs in the set not used for training.



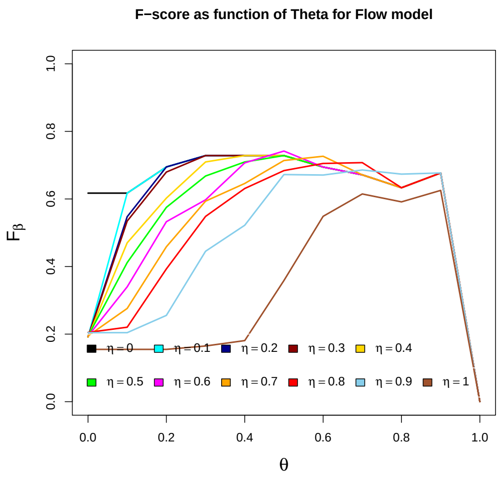
(a) Distance model when training on all datasets and validating on `data2_1`



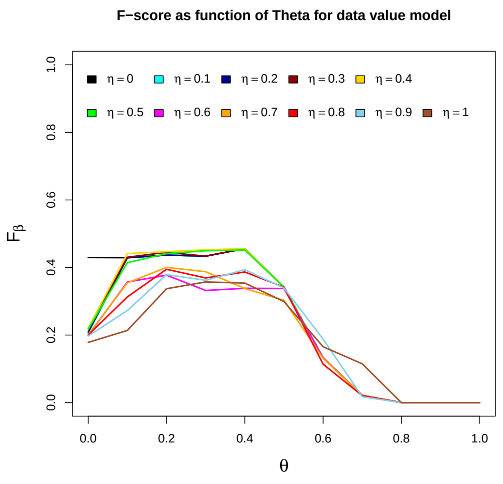
(b) Distance model when training on `data2_1` and validating on `data2_1`



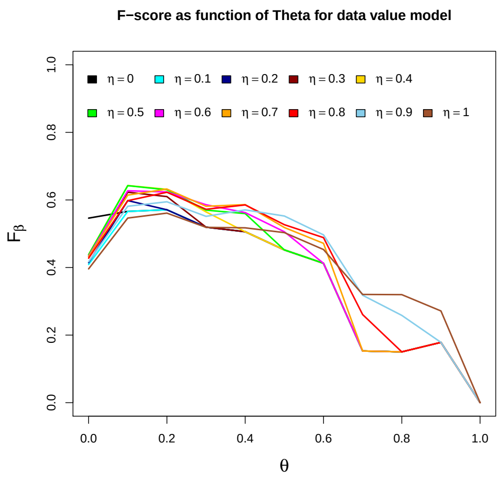
(c) Flow model when training on all datasets and validating on `data2_1`



(d) Flow model when training on `data2_1` and validating on `data2_1`

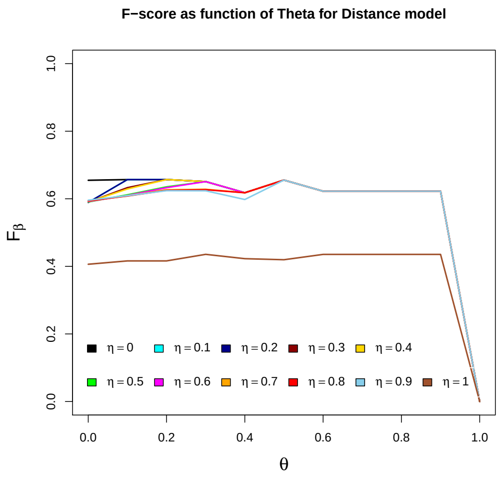


(e) Data value model when training on all datasets and validating on `data2_1`

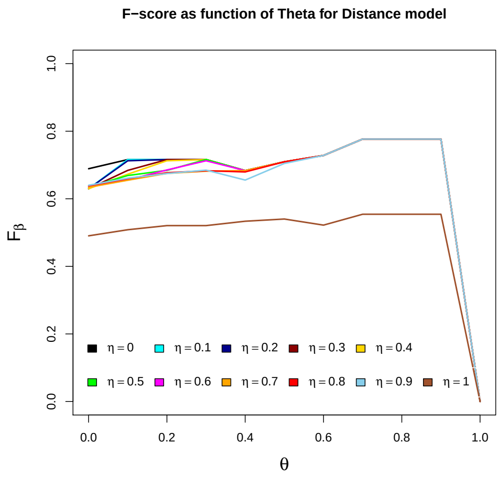


(f) Data value model when training on `data2_1` and validating on `data2_1`

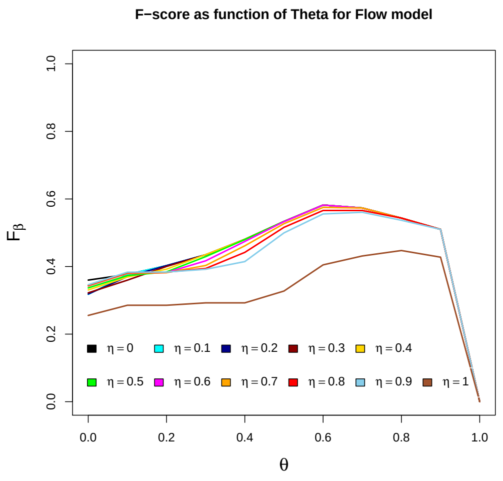
Figure 4.8: The Figure shows how the F_β-score changes depending on the parameters θ and η for 4.8(a) Distance model when all datasets is used for training, 4.8(b) Distance model when only `data2_1` is used for training, 4.8(c) Flow model when all datasets is used for training, 4.8(d) Flow model when only `data2_1` is used for training, and , 4.8(e) data value model when all datasets is used for training, and 4.8(f) when only `data2_1` is used for training. When training on all datasets, 100 random file pairs was sampled from each set. In the case of training only on `data2_1` one hundred file pairs was randomly sampled. In both cases 100 file pairs was randomly sampled for validation from `data2_1`. The validation was sampled from the remaining file pairs in the set not used for training.



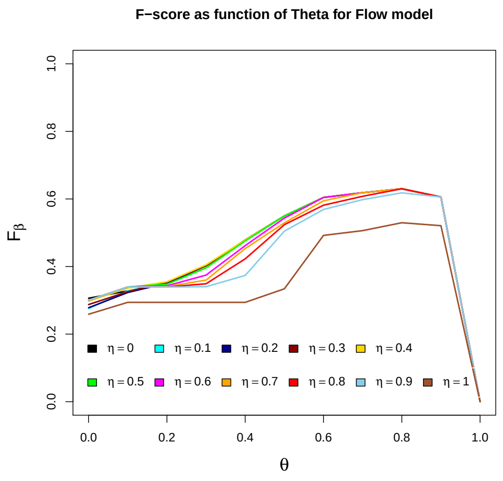
(a) Distance model when training on all datasets and validating on `data2_2`



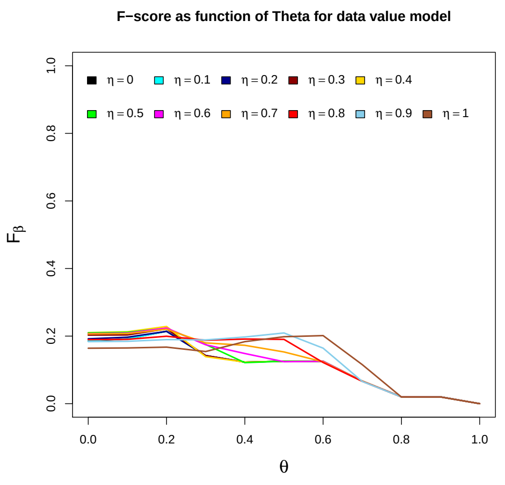
(b) Distance model when training on `data2_2` and validating on `data2_2`



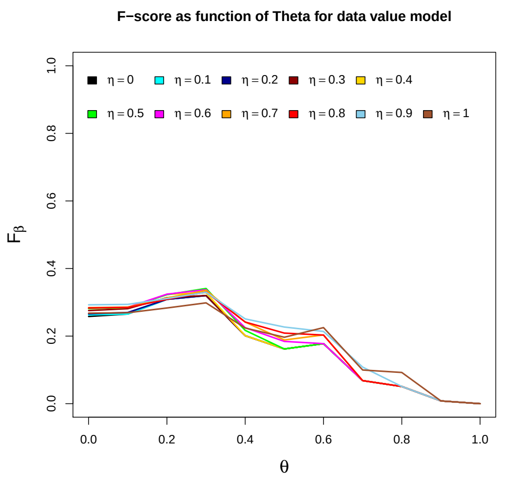
(c) Flow model when training on all datasets and validating on `data2_2`



(d) Flow model when training on `data2_2` and validating on `data2_2`

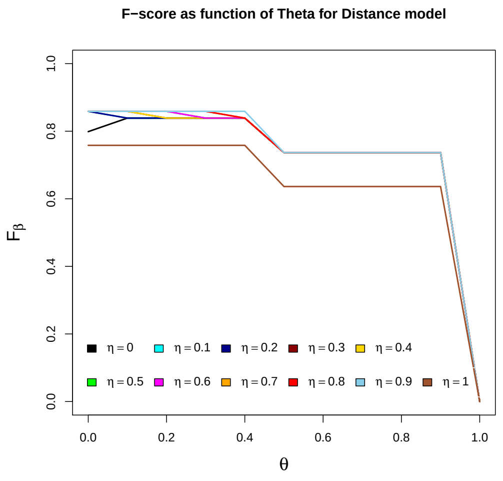


(e) Data value model when training on all datasets and validating on `data2_2`

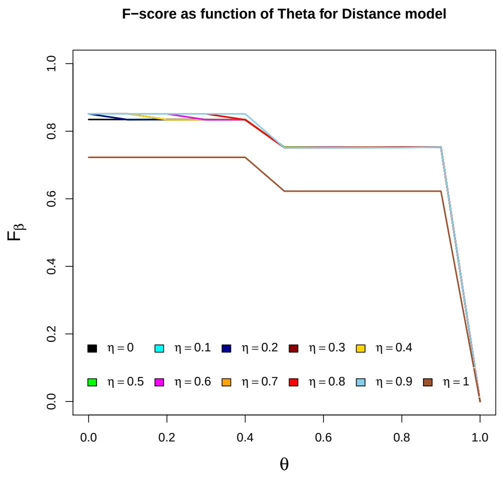


(f) Data value model when training on `data2_2` and validating on `data2_2`

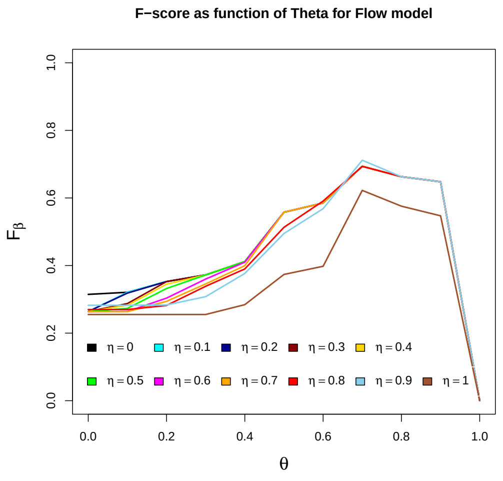
Figure 4.9: The Figure shows how the F_β-score changes depending on the parameters θ and η for 4.9(a) Distance model when all datasets is used for training, 4.9(b) Distance model when only `data2_2` is used for training, 4.9(c) Flow model when all datasets is used for training, 4.9(d) Flow model when only `data2_2` is used for training, and 4.9(e) data value model when all datasets is used for training, and 4.9(f) when only `data2_2` is used for training. When training on all datasets, 100 random file pairs was sampled from each set. In the case of training only on `data2_2` one hundred file pairs was randomly sampled. In both cases 100 file pairs was randomly sampled for validation from `data2_2`. The validation was sampled from the remaining file pairs in the set not used for training.



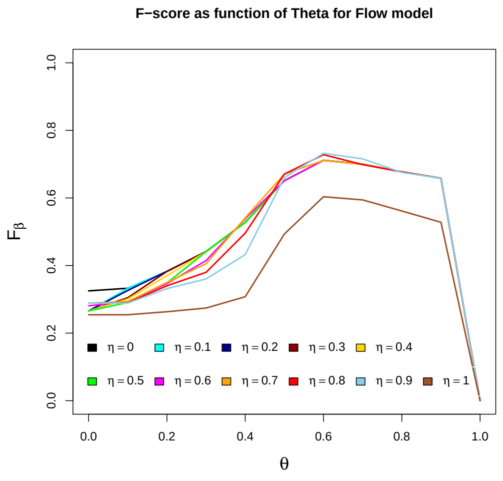
(a) Distance model when training on all datasets and validating on `data3_2`



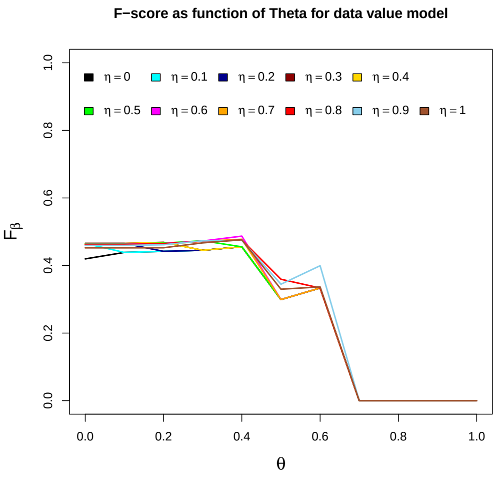
(b) Distance model when training on `data3_2` and validating on `data3_2`



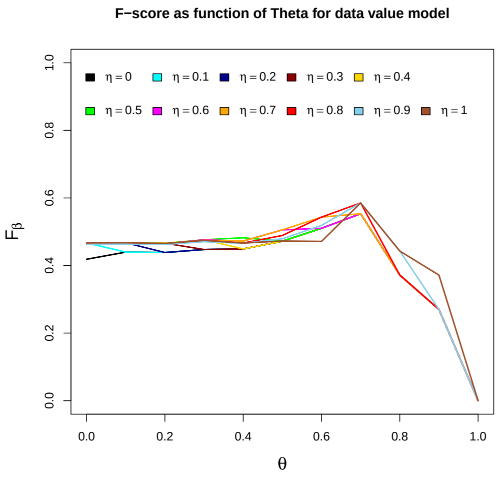
(c) Flow model when training on all datasets and validating on `data3_2`



(d) Flow model when training on `data3_2` and validating on `data3_2`



(e) Data value model when training on all datasets and validating on `data3_2`



(f) Data value model when training on `data3_1` and validating on `data3_2`

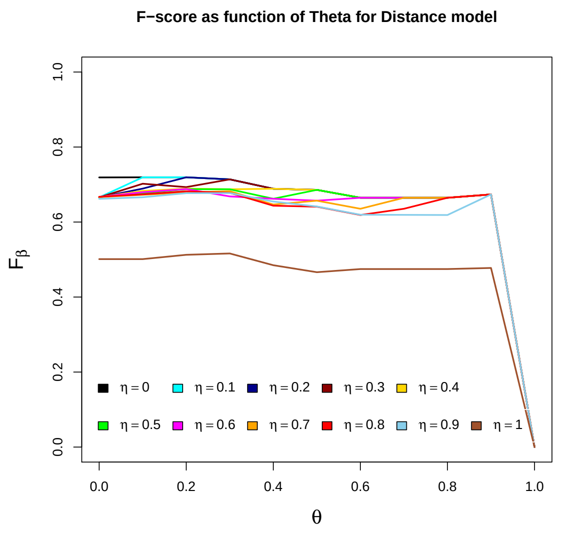
Figure 4.10: The Figure shows how the F_β-score changes depending on the parameters θ and η for 4.10(a) Distance model when all datasets is used for training, 4.10(b) Distance model when only `data3_2` is used for training, 4.10(c) Flow model when all datasets is used for training, 4.10(d) Flow model when only `data3_2` is used for training, and , 4.10(e) data value model when all datasets is used for training, and 4.10(f) when only `data3_2` is used for training. When training on all datasets, 100 random file pairs was sampled from each set. In the case of training only on `data3_2` one hundred file pairs was randomly sampled. In both cases 100 file pairs was randomly sampled for validation from `data3_2`. The validation was sampled from the remaining file pairs in the set not used for training.

4.2.4 Training and validating on all datasets

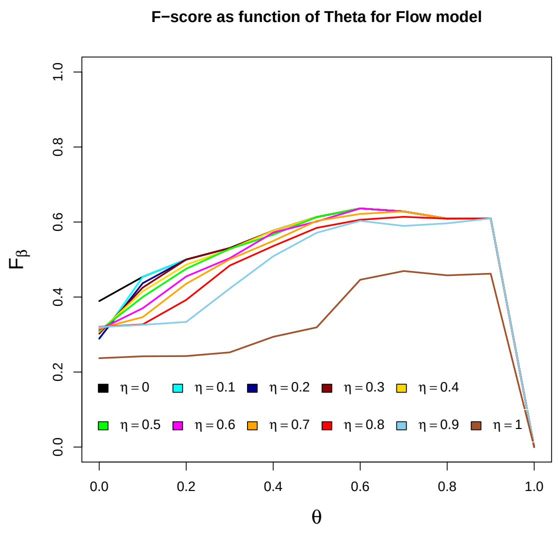
To examine how the models perform when all available datasets are used for training and all validation sets are used for validation, parameter sweeps are performed for this case. The resulting sweeps can be seen in Figure 4.11, and in Table 4.3 the highest achieved F_β-score is presented for the different models. The mean value of the achieved F_β-score over all datasets from Table 4.2 is computed to be 0.792 and 0.839 for the distance model when training on all datasets and validating on one, respectively training on one dataset and validating on the same one, 0.694 and 0.748 for the flow model when training on all datasets and validating on one, respectively training on one dataset and validating on the same one, and 0.407 and 0.532 or the data value model when training on all datasets and validating on one, respectively training on one dataset and validating on the same one. Comparing these values one can see that the achieved F_β-score when training on all data and validating on all data is lower than the mean achieved over all data sets when training on all datasets and validating on one at a time, and it is lower than all achieved F_β-scores when the models only are trained on the same set that it is validated on. This result is expected since the values of the parameters θ and η differs from set to set when the highest F_β-score is reached, which means that it is not possible to find values on the parameters θ and η in such a way that the optimal value is reached for all datasets simultaneously.

Model	θ	η	F_β**-score**
Distance	$0.2 \geq \theta \geq 0.1$	$\theta \geq \eta \geq 0$	0.719
Flow	0.6	$0.6 \geq \eta \geq 0$	0.636
Data Value	0.2	0.5	0.424

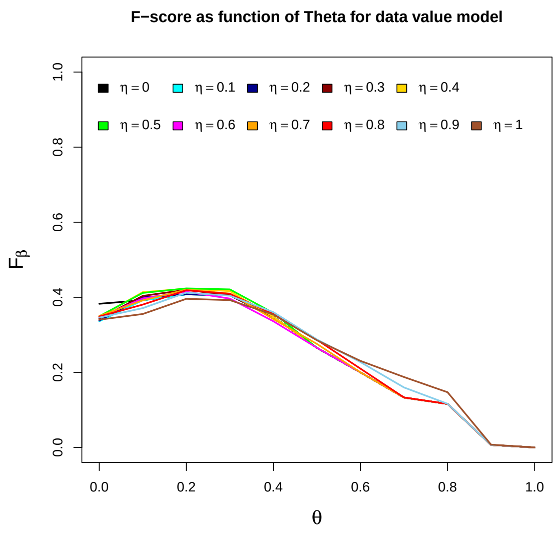
Table 4.3: The Table shows for which values of θ and η the optimal F_β-score is received, for β. The data is computed using 100 random sampled files from each available dataset for training, and 100 random sampled files from each available dataset for validation



(a) Parameter sweep for distance model



(b) Parameter sweep for flow model



(c) Parameter sweep for data value model

Figure 4.11: The Figure shows how the F_β-score changes for different values of the parameters θ and η. For this simulation 100 random file pairs was chosen from each of all five areas and used for training, and 100 file pairs from each area was used for validation.

4.2.5 Validation on different datasets than used for training

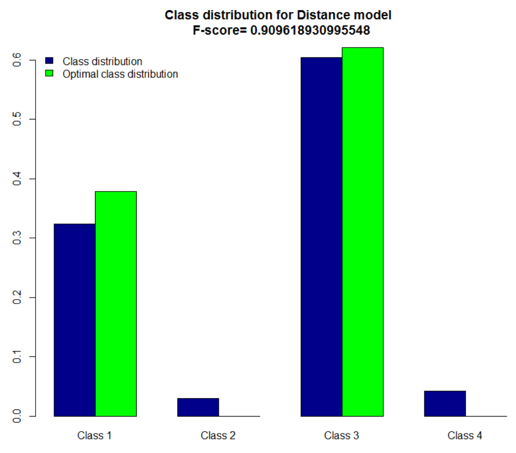
So far, the distance model has been better in all cases compared to the flow model, and the flow model is always better than the data value model. This is expected since the distance model achieves mapping probabilities only between whole paths that have been mapped before, and can in that way directly say if a path should be mapped or not, and gets it correct in most cases. The flow model is only looking at the words in the paths and can therefore get probabilities to map paths that have not been mapped earlier, if they contain words that have been mapped. The purpose of the flow model, was to be able to connect paths with each other even if they have not been seen before as long as the words contained by the path have been seen. To evaluate this function, the models are trained on all available datasets but one, and then evaluated on the dataset not included in the training part. This is done for each of the datasets, and the achieved result can be seen in Table 4.4. By looking at the resulting score for the different models and the different datasets, one can see that some of the datasets are not as different from each other as assumed. For the sets `data2_1` and `data2_2`, the distance model do find a lot of paths in the other sets during training that is identical to some paths in the validation set. In the other cases, one can see that for the dataset `data1_1`, no paths have been found in the training data that also exists in the validation set, but the flow model still succeeds to map a lot of paths correctly. When the set `data1_2` is used for validation, none of the models achieves a suitable score, but when `data3_2` or `data2_1` is used for validation, the flow model and the data value model achieves a similar score, which is interesting since this has not been seen in any other experiment. Even though the achieved score for the flow model is low in most cases in this simulation, it do show that the model can be of use when completely new data is to be mapped.

Validation set:	F_β-score distance model	F_β-score flow model	F_β-score Data value model
data1_1	0	0.702	0.342
data1_2	0	0.005	0.008
data2_1	0.494	0.437	0.435
data2_2	0.545	0.376	0.134
data3_2	0	0.245	0.261

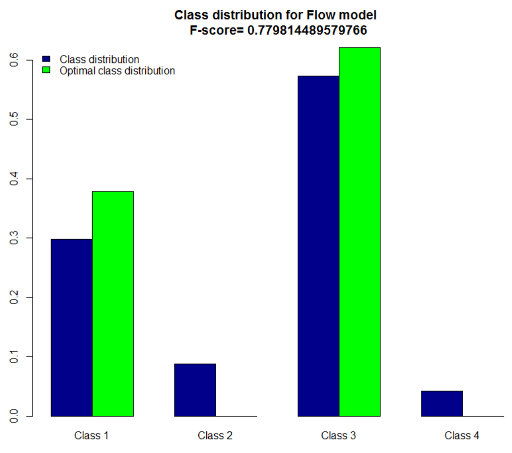
Table 4.4: Computing the F_β-score when files from one area is used for validation, and files from the rest for training. 100 files was sampled randomly from each area for training, and 100 files was sampled randomly for validation. The parameter values used was $\theta = 0.4$ for the distance and flow model, and $\theta = 0.2$ for the data value model. $\eta = 0.9$ was used for all models.
When the F_β-score is 0 for the distance model, this means that no paths in the validation set are recognized and hence nothing will be suggested for mapping. One can see that even though no paths is recognized, the flow model can still map paths, in some cases pretty accurate, only based on that the words is recognized in the data.

4.3 Class distribution

The different possible classifications of a particular mapping is defined in Table 2.3, were class 1 and class 3 are correct mappings, and class 2 and class 4 is incorrect mappings. As mentioned earlier, the F_β-score is used since all classes are not considered equal, but a class 2 error is considered worse than a class 4 error (see Section 2.3.1). In Figure 4.12 the class distribution is shown for the highest achieved F_β-score for dataset `data1_1`. As one can see, the largest classes are the correct ones, and for the distance model the class distribution is closer to the optimal distribution than the flow model, which is expected since the F_β-score is higher for that model. One can also see that using accuracy as score, the number of correct mappings divided by the total number of mappings, would give a result much higher than would be representative for the models. As one can see in Figure 4.12, about 60% of the paths should not be mapped. Knowing this, we could construct an algorithm that always decide not to map any path, and still receives an accuracy of 0.6 despite that algorithm being completely worthless in reality. For some datasets, the proportion of paths that should not be mapped can be even higher and that algorithm would in some cases receive an accuracy closer to 0.8, which definitely would not be a representative score of the models.
In Figure 4.13 the class distribution for the highest achieved F_β-score for dataset `data2_2` is presented. In this case, the achieved F_β-score is lower than for `data1_1`, and one can see in the figure how the classes have a different distribution. Class 1 is much farther away from the optimal value, and Class 4 is much larger than for `data1_1`.

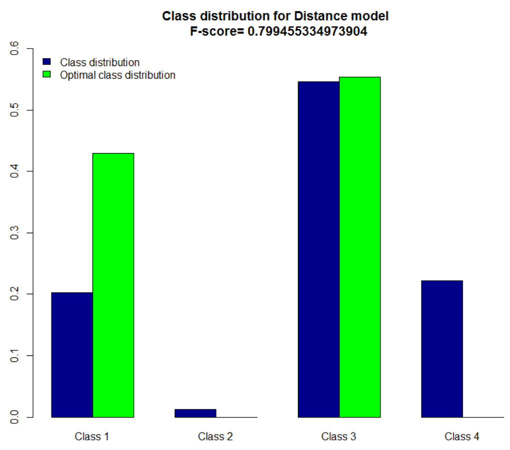


(a) Class distribution for distance model

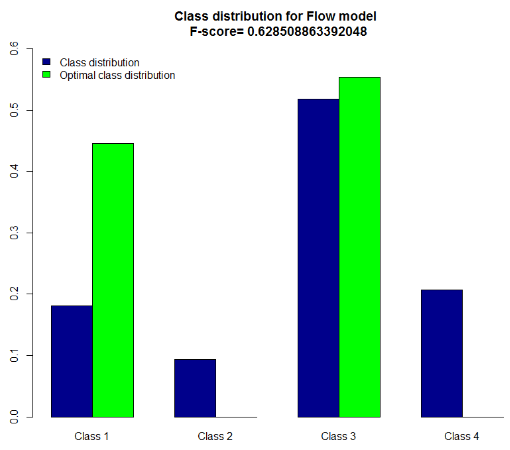


(b) Class distribution for flow model

Figure 4.12: The figure shows how the class distribution for the Distance model and the Flow model when the F_β-score achieved is 0.910 respectively 0.780 for the Distance model respectively the Flow model. The result is achieved when training on 100 randomly sampled files from dataset `data1_1` and validating on 100 randomly sampled files from the same dataset.



(a) Class distribution for distance model

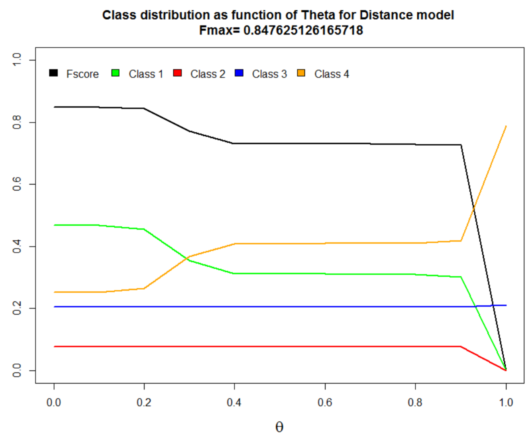


(b) Class distribution for flow model

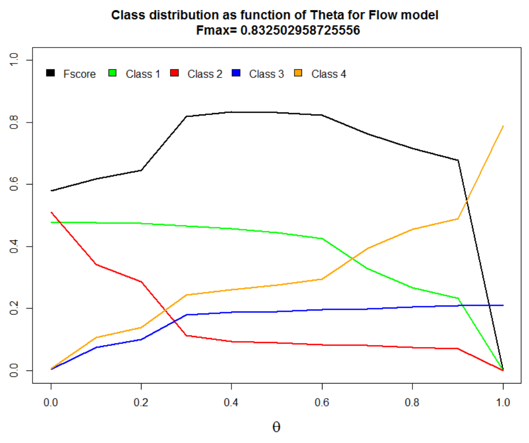
Figure 4.13: The figure shows how the class distribution for the Distance model and the Flow model when the F_β-score achieved is 0.799 respectively 0.629 for the Distance model respectively the Flow model. The result is achieved when training on 100 randomly sampled files from dataset `data2_2` and validating on 100 randomly sampled files from the same dataset.

In Figure 4.14 one can see how the class distribution changes when θ changes, together with the computed F_β-score. In Figure 4.14(a) one can see that the highest F_β-score is reached directly, and is decreasing when θ increases. One can see that Class 2 and Class 3 are almost constant for this model. Since Class 2 is consisting of faulty mappings, which in this case also receives a high enough F_β-score to avoid being filtered by θ, the hypothesis is that the specifications of what paths that should be mapped has changed. Since it is a possibility that two files map in some way at the start of a cooperation, but changes over time. One could for example decide that some of the data is not necessary to move, or it could be that the data did not have to be moved in the first place, but it did anyway. In these cases, possible matches for paths could still be included in that data format, but no data will be available in the validation files and hence generate an error, even though the semantic meaning of the actual mapped paths could be equivalent. Class 3 consists of paths that have not been mapped correctly, which indicates that the paths that is not mapping is paths that is either not recognized, or in different clusters and thereby gets a confidence level of exactly zero. The other two classes behave as one would think, when increasing the threshold some of the correct mapped paths will suddenly have a lower confidence level and be filtered away, and these mappings will instead be classified as Class 4, which is consisting of paths that has faulty been left unmapped. In Figure 4.14(b) the changes in the class distribution for the Flow model is presented, and one can see that they do change more than for the Distance model. First of all, it is clear that almost every path gets a non-zero confidence level to map towards at least one other path, since both Class 3 and Class 4 is zeros when $\theta = 0$. When θ increases, both the correct mappings and the faulty mappings (Class 1 and Class 2) decreases, but since Class 2 decreases faster that Class 1, the F_β-score increases.

For higher values of θ, Class 3 and Class 4 starts to behave like in the Distance model, and Class 2 is slowly decreasing until it reaches about the same level as in the optimal part of the Distance model.



(a) Class distribution for distance model as function of θ



(b) Class distribution for flow model as function of θ

Figure 4.14: The figure shows how the class distribution for the Distance model and the Flow model changes as function of θ, together with the F_β-score. As we can see in the figure, the best values for the Distance model is achieved at the very start, while the Flow model need to filter some faulty mappings using a higher threshold before reaching its optima.

5 Discussion and conclusions

In this chapter the results are discussed in a more general manner in order to put emphasis on the generic trends and predictive power of the models. We will draw conclusions regarding the quality of the model based on the earlier presented simulations, and conclude when and how they should be used to optimize their combined strength.
Lastly, ideas that were outside the scope of this project that could improve or extend the models are presented.

5.1 Discussion

Many aspects of this project made it hard to evaluate the actual quality of the models, and in many ways it was hard to create realistic environments using the data. The reasons for this are many, but mainly it falls back on a few key concepts. First of all, it has been problematic that the actual mapping specification has not been available, but only how the data has been organized in the file pairs. This has made it hard to both create the initial mapping specifications for training, but also to validate the suggested mapping from the models. It is known that faulty classifications of the suggested mappings have been made because of this, but in as many cases as possible the ambition has been to develop the scripts in such a way that the scoring of the models should be worse than they actually are, instead of the other way around. Second, since our knowledge regarding the files in the different sets of data has been limited, it has not been possible to divide file pairs in sets that in one way made sense, but have had to be satisfied by how they was divided in the first place. If more time had been available, it would have been interesting to create an unsupervised algorithm that looked for similarities in the file pairs and created clusters based on that concept. An approach like that would have given homogeneous sets that would differ from one another in a completely different way that they do now. And last, it would have been interesting to test the models on more data. The datasets that has been available have in many cases proven to be very much alike, and even though the number of file pairs has been large, the models can without any problem be trained on a small fraction and still perform very well. This may show some bias regarding the models, since if the validation set is almost identical to the training set, it's hard to draw conclusions of how the model would perform in reality.

5.1.1 Comparison of the models

The three models all have different strengths and weaknesses, The distance model performs best on earlier seen data formats as can be seen in Table 4.1, but is unable to recognize new formats. Table 4.4 shows the scores of the models when different sets are used for training and validation, and in the cases where the distance model gets a score greater than zero the formats are present in some of the other sets.
In most cases the flow model performs best on unseen data, and gets scores close to the distance model on earlier seen data. It gets the lowest score when validating on set `data1_2`, and this is because almost none of the words in the output files are present in the other sets.
Overall, the data value model gets lower score than the other models. This is mainly because it is unable to distinguish between similar data values when words in the paths are needed to find the correct map, a frequent error of the model is to mistake order dates for delivery dates and so on. It has the advantage over the other models that it only needs to recognize words in the output format and not the input format. This is especially useful when mapping new formats to the intermediate format, which is always the same in the available datasets. The results from the data value model can also be used in conjunction with the other models to determine if an output is compatible with the data connected to the output, and otherwise discard the mapping.

5.1.2 Scoring of the mapping

To be able to score the models an actual mapping has to be generated between the files. To do this a mapping algorithm has been developed, which always chooses the proposed mapping with the highest confidence level. In multiple situations several inputs can have high matching scores to the same output, but the correct one does not necessarily have the highest. The purpose of the project has not been to fully automate the process but to suggest a mapping, and can therefore list several matches for each output, and the expert can choose between these. As long as the correct mapping gets a high score, the models can be used in this way, but the procedure of mapping in the absence of an expert will affect the scoring in a negative way.

5.1.3 Parameter values

The optimal parameter values differ between the datasets (see Figures 4.1 through 4.5) but this is nothing critical since in reality the user can adjust these values for a given mapping until the initial suggestions are satisfactory. In general the distance model performs well with low thresholds θ which is due to the fact that earlier mapped paths probably should be mapped again, regardless of how often this was done in the training data. The thresholds for the data value model are strongly related to how the scaling factor λ was set. The parameter for lowering the confidence level of a path that has been suggested once before, η, does not affect the results much as long as it is smaller than one, which reflects the fact that each unique path should in most cases just be mapped once to make it possible for other nodes with the same path to be mapped.

5.1.4 Negative interference between the datasets

As can be seen in Table 4.2, the models usually perform worse when trained on all datasets and validated on one. This is because the data in the sets are quite homogeneous and gets over fitted to each specific set. This effect is larger on some of the sets, and negligible in some cases.

5.2 Conclusion

The models discussed in this report all reach the set goal, as they can suggest mappings and report a confidence level of these, but the accuracy of these mappings depends strongly on what model and what training set was used.
The models have different strengths, and if used beside each other a more sophisticated decision can be made by an expert. Given more training data from other systems the models can increase their score on unseen systems even more.
It is clear that statistics computed on different training data is needed for the models to perform well during mappings on different sorts of systems. This could be done in different ways, but in general many sets of mapping probabilities should be computed and when suggesting a mapping between two files, the set with the most common paths and words should be used.
It is also clear that a meta model that combines the three models should not just be some linear combination of them, but rather a combination of that and a state machine, were one can take into account that the distance model suggests mappings that probably are correct, and the flow model should be included when the paths are not known earlier. The data value model could both be used to check that the suggested mapping is trying to map data of the correct format to some path (most valuable for the flow model) and to help inserting new words into the model by using string comparison algorithms.

5.3 Future work

During the project we have had to delimit ourselves in order to meet the time constraint of the thesis work. Below some of the ideas that we considered outside the scope of this project are discussed.

5.3.1 Online training

All statistics used in the models are stored in such a way that they can be updated online. There are different ways to do this updating. The counts can simply be updated, but this would make the model inert to updates from small data sets. We believe that there still are some errors in the training data, and would therefore trust the feedback given by an expert more than the already existing training data. To avoid the inertia when updating the mapping probability one would instead want a process more like gradient descent, where the step length could be chosen in such a way that both inertia and unnecessary large overshoots is avoided.

5.3.2 Inserting new words into the flow model

New unseen words can be very similar to existing words. If words can be compared by similarity new words can be inserted into the word graph with edges connected to the similar words.
Words can also be split into several words if they are written in camel case or with some separator. These sub words could be used separately in the flow model.

5.3.3 Other initialization process for word mappings

In this project two different approaches for creating word mappings from path mappings have been tested, with different results. The first approach simply mapped each word in one path to each word in the other, which created a highly connected graph, instead of different clusters containing words with semantic equivalency, as wanted. The other approach made at most one connection between each word, and did so depending on where the words were in the path. The other approach worked better, mainly for the reason that the connected parts were now only between words that did not matter much for the mapping process. The most important words in the process are the last ones, since these words in more detail describe the data value connected to then, and are therefore more important to decide which other paths describe the same data. Other words then make the algorithm choose between equal paths. For example would a word like *ORDERHEAD* push the algorithm towards an equivalent group in the other format if it would be the case that a data existed in both the body and the head of the document.
When looking at the actual graph created in the initialization one can see that most of the words that frequently appear early in the paths belongs to the same cluster, even if they do not have any semantic connection, or even a functional connection. For example, if *ORDER* is the root element, it would be connected to other root elements, but because of the procedure of initializing word mappings, it would also be connected to later elements like *ORDERHEAD*, *ORDERLINE* and *ORDERFOOT*. To avoid this, another proposal for initializing the word mappings would be interesting to test. For example, instead of going from right to left when mapping words, one could go from outside in instead. For example, if the paths would be *ORDER/ORDERHEAD/Item/Qty* and *PIckingList/PickingHead/System/Product/Quantity*, the present mapping process would create the mappings “Quantity-Qty”, “Product-Item”, “System-ORDERHEAD”, “PickingHead-ORDER” and leave “ORDER” unmapped (for now). Since it is likely that in another case, the two root elements “PickingList” and “ORDER” will be mapped, Order will suddenly be connected to System, “PickingList” and “ORDERHEAD”, which is word that probably is not associated with the root element. If one instead went from outside and in, the created mappings would be “Quantity-Qty”, “PickingList-ORDER”, “Product-Item”, “PickingHead-ORDERHEAD” and leave “System” unmapped, which seems like a more reasonably mapping.

5.3.4 Using the structure of the XML-tree

If two leaf nodes have the same parent node in the input format, they should in most cases have the same parent in the output format. This can be crucial in some mappings to for example map the price and article number of the same article in an input format describing an order to the same group in the output format. This information has been discarded during this project as the main focus has been on associating paths with each other depending on the information collected from the semantic meaning of the paths itself, combined with the meanings of the words and the data formats connected to the path.

5.3.5 Extending the program to cover other formats

In this project we have worked exclusively with mappings between two XML-formats. There are many other standards used for data exchange and as long as one can extract descriptive words of the data types from the formats from some source the same principles can be used for mapping these formats. Interesting formats to investigate are, among others, XML-schemas, the EDIFACT-format and the JSON-format.

5.3.6 More advanced mapping algorithm

The scope of this project has mainly been focused on how one can associate paths in XML sheets with one another, and we have created three different models that does so. Little focus has been on the actual mapping process, even if some of the work has been done on that area. The main contributions on the actual mapping part has been to initiate a threshold to stop mapping when the confidence level is too low, and a parameter that reduces the confidence level for mapping a path twice. This has resulted in higher accuracy but more is to be done. For example, it is in the nature of integration to find relevant information in one format and transport that information to the correct place in another format. These formats are more or less place holders for the information, in such a way that another system can access the correct information and do the actual work. Because of this, it is unusual to have the same data in more than one place, although this happens, for some reasons. It could be that the data is not actually the same, in that sense that it just happens to have the same value, it could be that equal information is written in different ways, like an address where the city comes before the street at one place, and after in another. Using this information one could remove the possibility to map the same information twice and in that way decrease the number of incorrect mappings.

5.3.7 Determine when information should not be mapped

In this project, a threshold has been used to determine when one should not map two paths. This parameter has helped for two main reasons. In the distance model, it is unlikely that a path that gets a probability to be mapped, should not be so, unless there are errors in the training data, which there are. It has also helped regarding the parameter that decreasing the probability for mapping any path twice, even though the simulations concluded that it was best to never do so. Second it helped

in the flow model. Since the flow model gives a probability as long as some words in the flow have been mapped before, multiple paths that should not be associated with one another will get a flow between them since some of the less significant words still are associated with one another. Without the threshold, all paths containing data would be matched, where most matches would have a low confidence level ($\ll 1$) and hence probably be a faulty map.
Regardless of this, there are still multiple paths that the algorithm tries to map that should not be mapped, for a couple of reasons. The first is that in practice, data could have been mapped between two formats once, for some reason one decides that some data should not be transferred any more, but the data remain in the input format, and a possible match remains in the output format. If this case, and similar cases could be handled, the algorithm would be able to perform much better. When and why a paths should not be mapped in any given moment is not entirely clear, and more research need to be done regarding the specific cases, but it would still be interesting if a probability distribution for when one should not map any given path could be constructed.

Bibliography

[1] Philip A Bernstein, Jayant Madhavan, and Erhard Rahm. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11):695–701, 2011.

[2] Brigitte Bigi. *Using Kullback-Leibler distance for text categorization*. Springer, 2003.

[3] Carter T. Butts. *sna: Tools for Social Network Analysis*, 2014. R package version 2.3-2.

[4] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. *Ann Arbor MI*, 48113(2):161–175, 1994.

[5] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[6] AnHai Doan, Pedro Domingos, and Alon Y Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM Sigmod Record*, volume 30, pages 509–520. ACM, 2001.

[7] Ariyan Fazlollahi, Ulrik Franke, and Johan Ullberg. Benefits of enterprise integration: Review, classification, and suggestions for future research. In *Enterprise Interoperability*, pages 34–45. Springer, 2012.

[8] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *Advances in information retrieval*, pages 345–359. Springer, 2005.

[9] Kurt Hornik, Patrick Mair, Johannes Rauch, Wilhelm Geiger, Christian Buchta, and Ingo Feinerer. The textcat package for n-gram based text categorization in R. *Journal of Statistical Software*, 52(6):1–17, 2013.

[10] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[11] Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520, 2004.