

Topic Classification

A bachelor thesis by

EMILIE HELGESEN KARLSSON

138342823

TOBIAS BOLVIG RASMUSSEN BIELEFELDT

467050

Supervisor: Aisha Umair

Bachelor Project in Software Engineering

June 2021



UNIVERSITY OF
SOUTHERN DENMARK

The Maersk Mc-Kinney Moeller Institute

University of Southern Denmark

Abstract

Machine learning is a powerful tool which IT Relation is currently using to classify support emails to allow employees of the department specialising in the classified topic to answer the email directly. However the existing system is written in Python and with C# being what most of IT Relations developers specialise in it has lead to most if not all of IT Relations developers not wanting to touch the existing system. At the moment it is not an issue but as the company grows, gets more customers and increases the size of their departments IT Relation suspects it will become a problem in the near future. The project aims to create a maintainable system for IT Relation written in C# to classify their support emails by creating a machine learning model.

The project uses the ML.NET framework to create machine learning models capable of solving IT Relations multiclass classification problem while using design patterns to make the code as maintainable as possible. A facade pattern is used to make it easy to interact with the framework without having extended knowledge about ML.NET and a custom pre-processing is implemented to further enhance the usability and maintainability of the system. The project found using the One-vs-All method on a L-BFGS Logistic Regression algorithm the best at classifying IT Relation's emails. The projects final model was created and tested on 35 different categories with a total accuracy of 68.57% exceeding the existing system with more than 3%. This means the project not only managed to create a more accurate model but also a more maintainable by writing the system in the IT Relation's developers preferred language; C#.

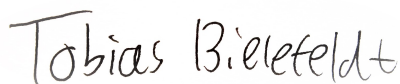
Preface

The project has been extremely beneficial for the group and has resulted in a much appreciated knowledge about machine learning and ML.NET. We believe machine learning to be an important part of the software industry, and more so each year. We are therefore glad this project gave us the opportunity to learn about machine learning and how topic classification is performed. Furthermore the project allowed us to code in C# and better our understanding of a coding language other than the main language taught as part of our Software Engineering degree. This was a great incentive for the project since it not only challenged us but also provides a useful new skill to take with us once we have finished our education. Lastly working with IT Relation meant we were dealing with a real case and a system which could potentially affect hundreds of people. This pushed us to keep fighting for the best possible outcome and to continuously learn no matter how tired or unmotivated we felt.

We would like to thank IT Relation A/S for the collaboration and allowing us to work with a real case. We would especially like to thank Thomas Køchs Nielsen for his insight into the company and the existing system and Kasper Axelsen Bolvig Hansen for his good ideas and help with translating IT Relation's database.

Secondly we would like to thank our supervisor, Aisha Umair, for the amazing guidance through out the entire project, her useful insight into a field which was very new to us and her creativity for finding new ways to handle problems we could not find ourselves.

Lastly we would like to thank the judges at the SDU Software Engineering day 2021 for awarding our project the prize for the best business potential.



Tobias Bolvig Rasmussen Bielefeldt
<tobie18@student.sdu.dk>



Emilie Helgesen Karlsson
<emkar18@student.sdu.dk>

Contents

Abstract	i
Preface	ii
List of Figures	vii
List of Tables	viii
List of Source Code	ix
Acronyms	x
Glossary	xi
1 Introduction	1
1.1 Motivation	1
1.2 Problem	1
1.3 Report Structure	2
2 Related Work	4
2.1 Prelude to ML.NET	7
2.1.1 ML.NET Workflow	7
3 Existing System	10
4 Methodology	12
4.1 Agile Development	12
4.2 Research	13
4.3 Requirement Prioritization	13
4.3.1 Stakeholders	13
4.4 Confusion Matrix and Evaluation	14
4.5 Testing and Continuous Integration	14
5 Requirements	15
5.1 Meetings with IT Relation	15
5.2 Functional Requirements	16
5.3 Non-Functional Requirements	17

5.3.1	Maintainability	17
5.3.2	Reliability	17
5.3.3	Performance	17
5.4	Project Scope and User Interaction	18
6	Analysis	19
6.1	Domain Model	19
6.2	Detailed Use Case	20
6.3	Sequence Diagram	21
7	Design	22
7.1	System Structure	22
7.1.1	Preprocessing Package	23
7.1.2	DatabaseCSV Package	24
7.1.3	MLModel Package	24
7.1.4	Translate Package	25
7.2	Design Patterns	25
7.2.1	Facade Pattern	25
7.2.2	Singleton Pattern	25
7.2.3	Builder Pattern	25
7.3	Patterns for Maintainability	26
7.4	Translation API	27
7.4.1	NTextcat	27
7.4.2	Implementation	28
8	Implementation	29
8.1	Implementation of Database Access	29
8.2	Auto Training with ML.NET	30
8.2.1	Implementation of Auto Training	31
8.3	Machine Learning Trainers	33
8.3.1	Logistic Regression	33
8.3.2	Binary Classification to Multiclass Classification	34
8.4	Model Training with ML.NET	35
8.5	Implementation of Custom Pre-processing	37
8.6	Implementation of ML.NET Pre-processing	38
9	Experimental Validation	40

9.1	Testing of Custom Pre-processing	40
9.1.1	Step 1) General Pre-processing	41
9.1.2	Step 2) Stop Words	42
9.1.3	Step 3) Word Normalization	42
9.2	Testing of ML.NET Pre-processing	44
9.2.1	Featurize Text	44
9.2.2	Bag of Word N-Grams	44
9.2.3	Bag of Char N-Grams	45
9.2.4	TF-IDF	46
9.2.5	Normalization	47
9.3	Evaluation of Pre-processing	47
9.4	Testing of Database Modification	48
10	Evaluating the Final Model	51
10.1	Comparing the System to the Existing System	53
11	Testing	56
11.1	Continuous Integration	56
11.2	Unit Testing	56
12	Discussion	59
12.1	Limited Knowledge about ML and ML.NET	59
12.2	Corrupt Data in IT Relations Database	59
12.3	Emails can belong to Multiple Categories	60
12.4	Pre-processing	60
12.5	The Use of Interfaces in the System	60
12.5.1	Danish and English Model	61
13	Conclusion	62
13.1	Future Work	63
13.1.1	Retrain	63
13.1.2	Multi-label Classification	63
13.1.3	Subcategory	63
	Appendices	67
A	BetterCodeHub Guidelines	67
B	Use Case Descriptions	68
C	Detailed Use Case: Train Model	69

D	SGD and L-BFGS	70
E	General Pre-processing: Explanations	72
F	Four Other General Pre-processing Methods	73
G	Confusion Matrix: LbfgsLogisticRegressionOvA	74
H	Confusion Matrix: SgdCalibratedOvA	77
I	Precision and Recall: LbfgsLogisticRegressionOvA	80
J	Precision and Recall: SgdCalibratedOvA	81
K	Unit Test: Pre-process Email	82

List of Figures

2.1	An example of how linear regression can misclassify	5
2.2	The ML.NET workflow	8
5.1	Use case diagram	18
6.1	Domain model	19
6.2	Sequence diagram: Train model	21
7.1	Package Diagram	22
7.2	The steps taken during the custom pre-processing	23
8.1	The overall approach to creating the systems model	29
8.2	The logistic function also called Sigmoid	34
8.3	How One-versus-All makes binary classification into multiclass classification	35
9.1	The three steps taken during the custom pre-processing	40
9.2	Final steps of the systems pre-processing	48
10.1	Comparing IT Relation's existing system to the projects system	54
1	Newtons Method	71

List of Tables

5.1	The functional requirements and their MoSCoW prioritization	16
6.1	Detailed Use Case: Train model	20
8.1	Trainers used in the system	31
9.1	General pre-processing results	41
9.2	Word List Results	42
9.3	Word normalization results	43
9.4	Results from testing the use of n-grams	45
9.5	Results from testing the use of character n-grams	46
9.6	Results from testing TF-IDF bag of words	47
9.7	Word List Results	47
9.8	Results from testing the entire pre-processing on the test data set	48
9.9	Test results from training and testing multiple trainers with a balanced data set	49
9.10	Results from testing different trainers on a unbalance database.	49
10.1	Precision and Recall of the final model	52

List of Source Code

1	The Systems main method instantiates a Database class and parse it to the ModelFacade	30
2	The ModelFacade uses the IDatabase to access the database .csv files . .	30
3	The <i>AutoCreateModel</i> method in ModelFacade	31
4	The <i>Create</i> method creating a auto model in ModelCreator	32
5	The <i>CreateNewPredictionEngine</i> method in ModelFacade	33
6	The <i>AddTrainerToPipeline</i> method in LbfgsOvaModelBuilder	35
7	Part of the <i>ManualCreateModel</i> method in ModelFacade	36
8	The <i>Create</i> method creating a manual model method in ModelCreator . .	37
9	The PreprocessingCustomAction class	38
10	The <i>GetTextFeaturizingOptions</i> method in AbstractModelBuilderManual	39
11	The <i>TestingCategoryClassification</i> unit test in MLUnitTest	57
12	The <i>TestingCustomPreprocessing</i> unit test in PreprocessorUnitTest . . .	82

Acronyms

CI - Continuous Integration

L-BFGS - Limited memory Broyden–Fletcher–Goldfarb–Shanno

ML - Machine Learning

OvA - One versus All

SGD - Stochastic Gradient Descent

SVM - Support Vector Machines

TF-IDF - Term Frequency - Inverse Document Frequency

Glossary

Agile Iterative Process - a team iterative process which differentiates from other iterative process models by making all larger decision at consensus, not focusing on specialisation among the team members and holding daily meetings.

Arrange, Act, Assert - A common pattern for arranging and formatting unit tests. The pattern is used to develop tests in tree steps: Arrange: setup the variables, Act: call the methods being tested and Assert: verify the result.

F-Score - A measurement of a binary classification models accuracy and is the mean of the precision and recall

False Negative - A false negative predictions occurs when a class is predicted to not belong to its actual class

False Positive - A prediction is a false positive if a class is predicted to belong to a class which is not its actual class.

Garbage in, garbage out - A concept in computer science stating that a programs results will be nonsense and if the input is too, regardless of how accurate its functionality is.

Monolingual - used to describe a text which is written in a single language.

Negative correlation - A statistical term describing the relationship between two variables. If one of the variables decreases so will the other and vice versa.

NuGet - In .NET developers share code via a package manager called NuGet. A NuGet package is a ZIP file with the .nupkg extension that contains compiled code (DLLs), and other files related to the code.

Scrum - a iterative lightweight process method consisting of three roles; Scrum Master, product owner and development team as well as specific events, artifacts and rules.

True Negative - A true negative occurs if a prediction finds that a class does not belong to a specific class that is not the actual class.

True Positive - A prediction is a true positive if the predicted class is the actual class, said with other words a correct prediction.

Unigram - Unigrams are a sequence of a single word. Likewise bigrams are sequences of two words.

1 | Introduction

This report details the making of a machine learning model which classifies the support emails IT Relation A/S receives from their customers. IT Relation A/S was founded in 2003 and is a Danish hosting business. They specialize in web hosting and IT outsourcing and work to optimize and solve their customers IT related problems. IT Relation is a large company with over 500 employees, a subsidiary company and partners in more than 55 countries.

The report will follow the iterative software development process from requirements to tests describing both the pre-processing and the classification of the data in the projects pursuit of achieving the best possible accuracy. Research will be heavily featured, since validating and experimenting with different approaches were essential for the project, as well as the process of ensuring a high maintainability in the outcome. The report concludes by discussing the problems with its findings and evaluating the outcome of this project.

1.1 | Motivation

IT Relation handles tasks related to hosting, outsourcing, security and service making their success depending on maintaining a reliable image by ensuring their users experience with IT goes as seamlessly as possible. Customers confidence in a service is an important asset for companies whose service entails supporting their customers after the initial purchase [3]. Since IT Relation customers perceive the companies service as their ability to extinguish IT related burdens it is important the process from a problem arises to it being fixed is as fast and reliable as possible.

The project is the result of a passive collaboration with IT Relation in the sense that the idea and certain requirements was presented by IT Relation but otherwise their only involvement with the project were in the form of guidance.

1.2 | Problem

Ideally IT Relations customers should be able to receive qualified answers right away due to a machine learning based classifier which can easily be maintained by IT Relations developers without taking up to much time. The existing system is not a sustainable solution for the developer team which have resulted in it decaying over time. Since the old machine learning model has not been sustained it can lead to more time being spend manually classifying email

as well as unnecessary time spend trying to understand the existing solution. Furthermore since IT Relation is a growing business it is important the solution has a high maintainability allowing for customization and changes as the needs arise. The project aims at developing a maintainable system which automatically classifies IT Relations emails into different topics allowing for the employee or department who specializes in that topic to answer it. This will make the process faster and give customers more qualified answers which will lead to an overall increase in customer support satisfaction.

1.3 | Report Structure

The report is intended to be read from start to finish, since that will explain the entire developing process. The following is a short structural overview of the coming chapters. It should be noted from this point all references to the systems classes will be in bold and system methods in cursive.

Chapter 2: Related Work - The research made regarding the subject to get a better understanding of advantages and disadvantages of different approaches.

Chapter 3: Existing System - Description of IT Relations existing system.

Chapter 4: Methodology - The tools and processes used during the project, this includes agile development, prioritization, confusion matrix, unit testing and continuous integration.

Chapter 5: Requirements - Explanation of the functional and non-functional requirements for the project and how they came to be. The system's user interactions are also included.

Chapter 6: Analysis - An overall view of what the system should be able to do. This includes a domain model, detailed use case and sequence diagram.

Chapter 7: Design - A description of the design decisions made during the project and why certain design patterns were used as well as an explanation of the translation API.

Chapter 8: Implementation - An explanation of the process of the implementation as well as describing the systems most important implementation.

Chapter 9: Experimental Validation - The results of testing different models and the steps taken to improve the models accuracy.

Chapter 10: Evaluating the Final Model - The evaluation of the systems final model. This includes a confusion matrix used to discuss results.

Chapter 11: Testing - The continuous integration and unit testing for the system is described.

Chapter 12: Discussion - A discussion of the flaws and strength of the project including but not limited to working with ML.NET and weaknesses of IT Relations database.

Chapter 13: Conclusion - Concluding the projects findings and outcome as well as explaining future work.

2 | Related Work

This chapter will discuss literature regarding text classification, to get an understanding of different approaches, as well as give an overview of ML.NET and its workflow since it is the framework used for the project.

The practice of classifying text has been around for many years, and has with time evolved from a manual approach where domain experts would classify text, to a rule-based system where queries of words determined the category and lastly with the advancements in machine learning; algorithms became the common practice to automatically classify text[10]. The following will take a look at the benefits and challenges of different approaches for text classification and how it have been handled by others.

There are primarily three main classification techniques; Statistical, Machine learning and Neural networks[20]. Chazdon, R. et al. created a statistical approach to classify generalists and specialists in habitats by using a multinomial model[5]. The probability model was used to estimate whether species are a generalist or specialist as well as minimize bias due to insufficient entries. The research found the method was competitive with non statistical classifiers and even classified more bird species. However since statistical approaches are generally based around a probability model and are mainly used by statisticians it was not used in this project.

Neural networks are inspired by the human brain and consists of layers of nodes connecting with each other to form a non-linear function. Each node can be seen as its own linear regression model that uses an activation function to map the outputs to either 1 or 0. The neural networks approach has previously been used by Wang Yawen et al. for spam filtering[30]. The research constructed a deep neural network with two hidden layers with the aim of proving the effect of spam filtering with the neural network approach. The research found the approach highly effective with the deep neural network outperforming the Naive Bayes algorithm. Neural networks is however not implemented in the ML.NET framework and the approach was therefore not used for this project.

The last approach of using machine learning functions by learning a task based on given data, since this is the approach chosen for the project the remainder of this section will look into different algorithms and tools for natural language processing using machine learning.

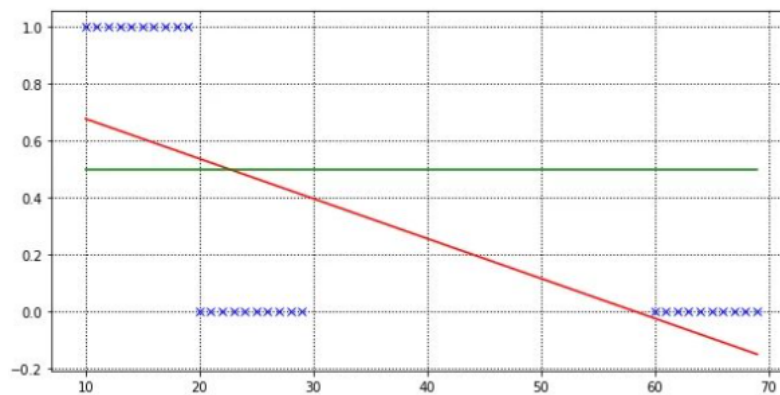
One of the simplest approaches of machine learning is Linear regression. The goal of linear regression is to fit a linear function $f(x)$ to predict an output y with features x . With only one feature it is written as shown below with w being the weights that a machine learning algorithm will manipulate in order to improve the functions accuracy:

$$y = f(x) = w_0 + x * w_1 \quad \text{or with n features:} \quad f(x) = w_0 + x_1 * w_1 + x_2 * w_2 + \dots x_n * w_n$$

If a dummy feature is added for the value $x_0 = 1$ the above function is written as:

$$f(x) = \sum_{i=0}^n w_i * x_i$$

Classification problems do however need the output y to be 1 or 0, which this function does not guarantee. One could map every prediction below 0.5 to 0 and above to 1 and for some very balanced data sets than can work, but if the data that is used is not balanced, the value x where $f(x) = 0.5$ will not be in the theoretical correct place and the function will be biased towards the outliers which causes the model to make bad predictions. In figure 2.1 the red line is the linear regression function $f(x)$ which is trained on the 'x' marks in the figure. The green line is $y = 0.5$. If the regression function $f(x)$ were used with the x values from $x = 20$ to 22 the outputs would be more than 0.5 which means this algorithm would classify them as being part of class 1 but as seen in the figure they are part of the class 0, which means the algorithm would be incorrect.



Source: [11]

Figure 2.1: An example of how linear regression can misclassify

For this reason research shows that linear regression is not suitable for classification problems [11] and it will therefore not be used in this project.

During the project multiple different methods were researched, one of them being Pairwise Coupling. Pairwise Coupling is a method used to modify a binary classification algorithm to be used as a multi-class classification algorithm. Pöyhönen, S. et al. used this approach to research if a binary SVM-based classifier used with pairwise coupling would produce sufficient result[21]. The research found that a SVM-based classifier could be successfully used with pairwise coupling to solve multiclass classification problems in an induction motor.

Pairwise coupling create a binary classification model on each possible pair of classes. So if there were three classes, 0, 1 and 2, a binary classification model would be created for the pairs (0,0), (0,1), (0,2), (1,1), (1,2), and (2,2). Since the project has a total of 176 categories out of which 35 are used it would result in 630 pairs¹ needed and a binary classification model would have to be created and trained for each. For this reason pairwise coupling is not included in this project since it would be to time consuming and since the previously mentioned research by Pöyhönen, S. et al. does not support it performing better than other available solutions.

As a part of the text pre-processing word normalization will be used. One form of word normalization which will be looked into is stemming. There exist many different implementations of stemming but they can generally be grouped into two different kinds; truncating and statistical. Truncating stemming is where suffixes are removed from words using predefined rules, of which the most popular type is the Porter stemmer[8]. Statistic stemming is where suffixes are removed based on statistical analysis and techniques, such as the N-Gram stemmer which is build on the assumption that words stemmed to the same stem has a higher proportion of an identical character n-grams. A research paper by Majumder, P. et al. looks into the difference between statistical and truncating stemming by comparing their affect on information retrieval[15]. After using implementations of both kinds of stemmer during a Monolingual² information retrieval run for French the paper found the statistical stemmer was comparable, but not better than the truncating stemmer. For this project it was not possible to find an implementation of a statistical stemmer in C# and taking the papers results into consideration it was decided it would be sufficient to only focus on truncate stemming.

¹Calculated by using the formula: $\frac{n(n+1)}{2}$ with n = 35

²See glossary for an explanation of "Monolingual"

2.1 | Prelude to ML.NET

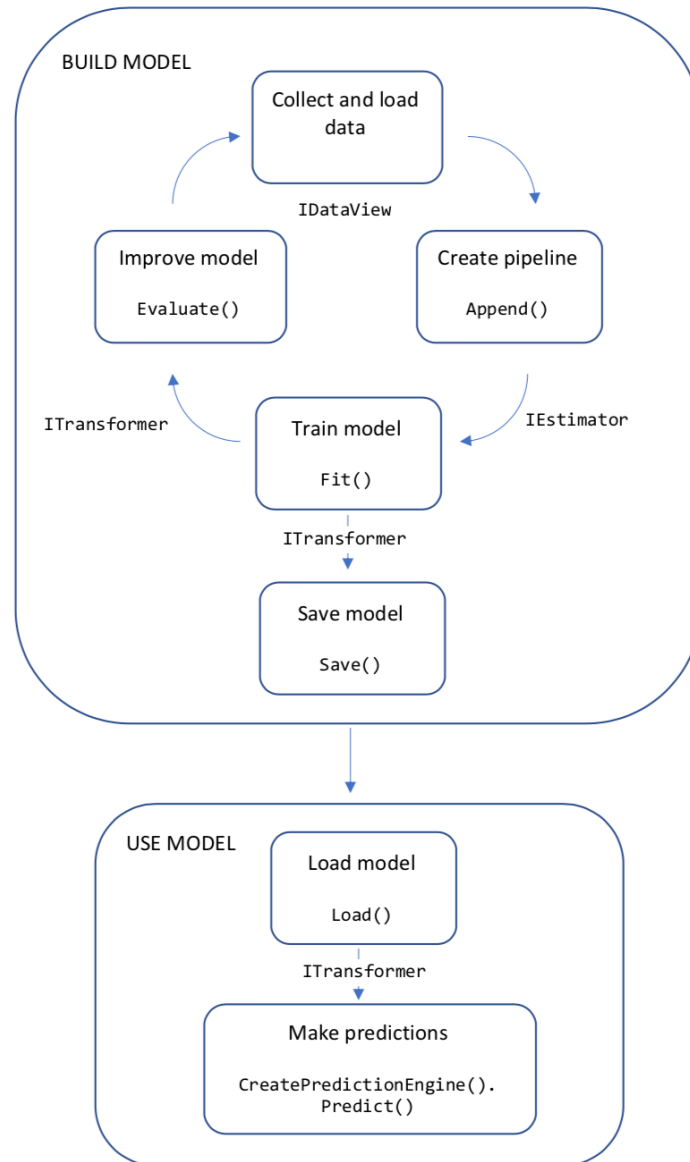
In this project the ML.NET framework was used. This section will give a short overview of ML.NET but will not go in-depth on what it is, but rather why it was used.

ML.NET is an open source, and cross platform machine learning framework for the .NET developer platform[17]. It was initially released in 2018 and can be used with both the C# and F# programming languages. The fact that it was developed by Microsoft with C# developers in mind made it the obvious choice for this project since IT Relations biggest request was for the solution to be easily understandable for their developers who write in C#. Furthermore ML.NET is the dominant machine learning framework for C# development and even though it is fairly new it already has many different features and great documentation. Another reason why ML.NET was chosen has to do with the ML.NET model builder which provides a visual interface to build and train models. This was especially useful in the beginning of the project to get an understanding of how ML.NET functions and test a few scenarios without being too time consuming.

Besides ML.NET, Microsoft also offers two other machine learning services; Azure Cognitive Services and Azure Machine Learning. Azure Cognitive Services provides an API with a pre-built machine learning models. It is easy to implement and does not require extended knowledge about machine learning. Azure Machine Learning on the other hand provides an environment on which to host machine learning models. It also provides automated machine learning and has a drag and drop interface limiting the coding knowledge required. Neither of the services however allows for customization of the solution nor integrates as easily with .NET applications as ML.NET does[17] and it was therefore decided to not include them in the project. The Azure Cognitive service was however used to translate the data which will be explained in section 7.4.

2.1.1 ML.NET Workflow

The ML.NET workflow, seen on figure 2.2 will be described in the following section as well as the three main interfaces; IDataView, ITransformer and IEstimator. The three interfaces are the main interfaces ML.NET is based on and how they interact is important to understand the ML.NET workflow and will therefore be explained.



Source: [17]

Figure 2.2: The ML.NET workflow

IDataView: In ML.NET the IDataView classes are simple classes that provides processing of schematized data. A schema holds information about what columns there are and what type they are. The rows are not saved in the DataView class, but it holds information about where the data is and what it is.

ITransformer: The ITransformer classes take data in the form of a DataView and transforms it. The transformation can be simple pre-proccsing or it can be more complex such as

a machine learning model. A model is considered a transformer since making a prediction in ML.NET is also considered a transformation, as it transforms one `DataView` to another, in the project an email's text to a `categoryID`. A useful feature of the Transformer is the ability to link transformers together and the output will still be one Transformer. This means a Transformer is able to do multiple transformations after each other as a chain, so it could transform a `DataView` via a pre-processing transformer and then predict what category it belongs to using a machine learning model transformer. `ITransformers` can also be saved and loaded as zip files.

To use a transformer to make a prediction it is possible to create a `DataView` of only 1 instance but a simpler solution is to use the `PredictionEngine`. When a `PredictionEngine` is created it will take an output schema, an input schema and a Transformer whose input and output fit the one given to the `PredictionEngine`. When the *Predict* method is called it will simply output the prediction.

IEstimator: The `IEstimator` classes are untrained transformers and are used to define what transformations a Transformer class should be able to do, which includes machine learning algorithms. When the *Fit(DataView)* method is called it will fit the untrained transformer to the `DataView` which will create a Transformer designed to transform that data. During this process it will train any machine learning algorithms that are on the estimator. Just as transformers, estimators can be linked together. When the *Fit(DataView)* method is called on an Estimator class with multiple estimators the result will still only be one transformer but the transformer will do all the transformations defined in the estimators.

The workflow of ML.NET is to load data with the **IDataview** this can be done from files or databases. Then a `IEstimator` pipeline is created. The pipeline defines what transformations should happen to the data and includes the machine learning predictions as that is also just a transformation. The *Fit(DataView)* method is used to create an **ITransformer** from the pipeline, this also trains any machine learning algorithms on the pipeline. When the Transformer is created the machine learning model has been created inside the transformer and it is possible to evaluate the model and make changes to the **IDataView** or **IEstimator**. Or, if the model is sufficient to save it.

3 | Existing System

At the beginning of the project, the code of IT Relations existing system was made available. Unfortunately IT Relation generally do not use documentation and had therefore none for their existing system. This section will therefore describe the existing system with the information gathered from examining the code and the meetings detailed in section 5.1.

IT Relations current solution is written in Python using the Scikit-Learn library together with Jupyter Notebook. The existing system can be split up into 3 subsystems:

1) The first subsystem creates models.

The first system downloads all the incidents and then pre-process them. The pre-processing is done in steps, to begin with it removes "Mail message from...", which is a prefix their ticket system adds to emails. It then removes email signatures and endings such as "Best regards" and "Warm regards", which also removes everything after. Everything is then parsed to lower case and if the ticket is not in English it is translated using Azure cognitive services, afterwards stop words are removed and the remaining words are stemmed and punctuation is removed. It then splits the remaining email up using bag of words and score them using term frequency inverse document frequency (TF-IDF). Lastly it trains models. The algorithms the existing system uses are: Stochastic Gradient Descent(SGD), Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) Logistic Regression, Linear Support Vector, Random Forest, Multinomial Naive Bayes and Multi-Layer Perceptron. Models are created using all of the algorithms, but only the best is saved which in their case was the L-BFGS Logistic Regression. This Subsystem does not upload anything to azure cloud which means this is mainly used for manual creation and testing of models.

2) The second subsystem loads a model and use it to make predictions.

This subsystem loads the model saved by subsystem 1 or 3 and when an email is received it pre-process the email and use the loaded model to make a prediction.

3) The third subsystem retrains models and uploads them to Azure cloud.

In their system "retraining" a model means to create and train a new one. It does the same as in system 1 with the main difference being how subsystem 3 uploads the model to the Azure cloud automatically whereas subsystem 1 only creates it.

During the meetings with IT Relation it was mentioned that their existing system had an accuracy of about 80%. Since the code was written using Jupyter Notebook the results were saved and available, the highest model found was a SGD model with an accuracy of 78%. However that model seemed to be a either a test model or a very early model because it only categorised on 7 categories. A more accurate representation of the existing system found in the code was a L-BFGS Logistic Regression model which predicted on 35 categories and had a accuracy of 65%. This model was inside the retrain subsystem and is therefore believed to be the newest. Further supporting this theory is the models name "LR_20191028.pkl" which stands for Logistic Regression 28th of October 2019 which is the newest date among the saved models in the code.

4 | Methodology

This chapter will go through the work methodology for the project, by explaining the approaches and tools used. It will however not be an in-depth description on what they are, but rather why they were used.

4.1 | Agile Development

Throughout the project the group made use of an agile development process. An agile process allows for greater changes in design due to its short iterations[29, p.84] and was therefore deemed necessary due to the groups minimum experience with machine learning.

It is widely believed among engineers that a good process results in a good product[22, p.179] because of this the process model Agile Iterative Process (AIP)¹, and more precisely Scrum² was chosen as the process model for the project.

Scrum was chosen over other methodologies such as Unified Process due to it being a less heavyweight methodology allowing for more time on development rather than documenting artifacts as well as being more flexible when it comes to deciding on big design decisions[2, p.94], while still offering more structure than the Kanban Method which primarily works on the just-in-time principle[2, p.96].

Since scrum and AIP's in general are designed for teams of roughly 5-10 members and this group only consisted of two it was not possible to incorporate every element of Scrum. This meant the group did not have a scrum master and instead both members worked to ensure the process was kept which also aligned with general decision making which was made by consensus. The role of Product Owner were to an extent IT Relation, since they contributed to ensure the project meet the objectives and goals. Otherwise daily meetings were held as well as reviews of sprints to ensure the quality of the system as well as the project not falling behind on schedule. Another important aspect incorporated from Scrum was the backlog, which gave an overview of the projects tasks and ensured an iterative process.

¹See Glossary for an explanation of 'Agile Iterative Process'.

²See Glossary for an explanation of 'Scrum'.

4.2 | Research

Very early on in the project it became clear the project to some extent would resemble a research paper rather than the software development processes the group had prior experience with. This was the case, since the group had limited knowledge about machine learning and natural language processing before hand which meant the majority of the time on the project was spend researching. It was prioritized to try varying different methods the group came across during the research to be able to analyse what approach would be the most beneficial for the system. This was especially the case for the pre-processing where different approaches such as stemming and lemmatization were analysed, tested and evaluated in order to be able to get an in-depth interpretation and base decisions on what to use on informed opinions. This was also the case when it came to picking the best algorithm for the machine learning model where multiple were tried in order to ensure the best result. The prioritization of researching different approaches and methods were a necessity for the project, due to the groups limited knowledge. Because of this chapter 9 and 10 was prioritized to entail the research from the entire process in order to give a clear understanding of how the solution came to be.

4.3 | Requirement Prioritization

A different aspect of prioritization has to do with requirement prioritization. For the project it was chosen to use the MoSCoW³ technique. The solution is a relatively small system and it was therefore not deemed necessary to use more advanced methods such as RUP[2, p. 60]. This choice did however mean the project suffered by the MoSCoW's techniques inabilities to take precedence into consideration, which meant that requirements was not prioritized based on how much time they would take to complete nor how much research was needed before hand, but simply on their importance. Using the MoSCoW technique was however still the right decision for the project, since its simplicity was not only time efficient, but made it easily understandable for the stakeholders and therefore easy to ensure the projects requirement prioritization aligned with the stakeholders.

4.3.1 Stakeholders

It is always important to identify the stakeholders and their viewpoint when working on a project. The projects main stakeholder is IT Relation. The project aims to develop a

³**M**ust have, **S**hould have, **C**ould have, **W**on't have.

solution which will benefit IT Relation and their viewpoint are therefore necessary to take into consideration.

IT Relation has the role of being the customers, users and resource managers for the project, since they are the projects intended recipient as well as providers of the data the project is catered around. IT Relations customers would also benefit from this solution since it aims to make their interaction with IT Relations customer support more efficient and reliable, but since they will never interact directly with the system their viewpoints will not be taken into consideration.

4.4 | Confusion Matrix and Evaluation

Predictions in a machine learning model has four outcomes; a true positive⁴, true negative⁵, false positive⁶ or a false negative⁷. These are often illustrated in a confusion matrix which shows the counts of predicted and actual classes. the project used the ML.NET framework to configure confusion matrices. The outcome of the matrices can be used to evaluate a model based on accuracy, precision and recall. The project has mainly focused on accuracy, but will discuss all three in chapter 10.

4.5 | Testing and Continuous Integration

To validate the system, testing will be introduced in the form of unit testing. By testing units of code it is ensured the most important units in the system delivers as expected. Continuous Integration (CI) is a practice that automates the process of validation before integrating new code with existing code. CI works seamlessly with agile development and was used in the project by creating a CI Pipeline using GitHub Actions. This was done with the intent that CI makes the system more maintainable, since it will be harder to add code which might break other parts of the system.

⁴See glossary for an explanation of "True Positive"

⁵See glossary for an explanation of "True Negative"

⁶See glossary for an explanation of "False Positive"

⁷See glossary for an explanation of "False Negative"

5 | Requirements

In any software development project it is important to find and analyse what the system should do, which results in the high-level specification that is the purpose of requirements engineering [2, p.51]. This chapter will describe the projects requirements and how they came to be. It will also cover both the reasoning for them and their prioritization.

5.1 | Meetings with IT Relation

To obtain an initial set of requirements meetings were held with Thomas Køchs Nielsen and Kasper Axelsen Bolvig Hansen from IT Relation at the beginning of the project. The meetings were used to get an understanding of IT Relation as a company and their existing system. Besides gathering fundamental information the meetings had the underlying purpose of not only getting an insight into Mr. Nielsen and Mr. Hansen's perspective and ideas on the project, but to gather requirements and an understanding of the problems with their existing system.

From the meetings it became clear that seen from their perspective the project needed to address 2 problems. The first and most important relating to how their current development team uses C# as their programming language while the existing system is written in Python. Mr. Nielsen described IT Relation as being a "Microsoft oriented company", in the sense that they primarily uses the .NET platform for development. Because of this they were interested in looking into the possibility of switching to the ML.NET framework. Even though the existing system functions, the fact that, it is written in Python makes it less maintainable which is a concern of theirs especially since it is not a piece of code they often look at.

The second problem is the pre-processing. The existing system does, as mentioned earlier, pre-process the emails, but it is still a problem the project needs to be aware of since the ticketing system and auto monitoring systems used by IT Relation adds unnecessary text to the emails.

Another important piece of information gathered from the meetings was the structure of the database. Every email is classified by service, category and subcategory, with each service having a couple of categories which then in turn has a few subcategories. Mr. Hansen explained that they rarely use the subcategory and did not expect the projects system to

be able to predict it. Because of this he also mentioned that it might be easier to try and predict the category rather than first the service and then the category. Since the category would automatically determine the service it could potentially result in a higher accuracy.

From the information gathered at the meetings with IT Relation it became clear maintainability was their biggest non-functional priority which to them mainly meant the system should be implemented in C# which therefore should be part of the projects requirements. Furthermore it would be crucial to look into pre-processing to ensure the project did not end up in a 'garbage in, garbage out'¹ scenario and lastly how it was recommended the system should only try and predict the category.

5.2 | Functional Requirements

The meetings with IT Relation helped identify the most important functional requirements with the rest being found by looking at IT Relations database and their existing system to get a more detailed view and understanding of what behavior the system should offer. The final description of the functional requirements can be seen along with their MoSCoW² prioritization in table 5.1.

ID	Requirement	MoSCoW
F001	System is written in C#	M
F002	System can classify emails according to their category	M
F003	Emails can be pre-processed to remove noise	M
F004	Solution is build on the ML.NET framework	S
F005	System can classify emails according to their service	C
F006	Classified emails should be saved to IT Relations database	W

Table 5.1: The functional requirements and their MoSCoW prioritization

Once the functional requirements were determined they were shown to Mr. Nielsen and Mr. Hansen from IT Relation to ensure an understanding of the problem were reached and the ideas of what the system should do aligned, which was confirmed. Overall the project does not have many requirements, since the problem does not need to offer much behavior. The majority of them are however in the Must have category since IT Relation had very few but important requests. Requirement F006 was prioritized as a Wont have, since an integration

¹See Glossary for an explanation of 'Garbage in, garbage out'.

²Must have, Should have, Could have, Won't have. See glossary for a further explanation of 'MoSCoW'.

with IT Relations system was deemed outside of the projects scope. Requirement F005 is also not a part of the system, since a satisfying accuracy was reach with classifying by category which was also what IT Relation had requested be done.

5.3 | Non-Functional Requirements

Non-functional requirements was also discussed for the project, since it was important to set constraints on the system in general and not just its functionality, to ensure it delivered as expected. The following are the final non-functional requirements for the project.

5.3.1 Maintainability

Even though using the ML.NET framework was not a direct requirement from IT Relation and merely a suggestion, it was decided early on in the project; it would be unwise to not use the framework. The framework had great documentation which made development less time consuming and would make the system more maintainable for IT Relation. More importantly the framework had performed well with large data-sets[1] and is built for .NET developers like the developer team at IT Relation. It was therefore deemed the best option. Furthermore the four guidelines; 1) Write Short Units of Code, 2) Write Simple Units of Code, 3) Write Code Once, and 10) Write Clean Code from BetterCodeHub.com³ will be used to minimize code smells which will in turn make the system more maintainable.

5.3.2 Reliability

As mentioned in chapter 3 Mr. Hansen and Mr. Nielsen believed their existing system had an accuracy of 80% even though they did not have any tests to back it up. However, 65% was the highest found in the existing systems code. The reason Mr. Hansen and Mr. Nielsen believed their system to be 80% accurate is discussed in section 12.3. To ensure the reliability of the system an accuracy of at least 65% will be required.

5.3.3 Performance

In terms of performance it was determined it should be possible to train the systems model relatively fast, meaning it should be possible within 4 hours to ensure it is manageable within a single workday.

³See Appendix A for a description of the guidelines

5.4 | Project Scope and User Interaction

The requirements listed in table 5.1 are based on the final outcome of the system where it will be integrated into IT Relations system and new emails when received will automatically be classified and sent to the correct employee. The integration of the system was however deemed out of the scope for the project, since it is not required in order for the project to showcase how to classify text in C# nor the aspect of developing a maintainable system.

Because of the projects scope it was decided to add some user interaction to the system. These interactions both allow the system to be tested and used without being integrated, but they will also allow IT Relations developers to train a new model once it is integrated, which also adds to the maintainability.

The most important user interactions are shown in the use case diagram in figure 5.1. A list describing all interactions can be seen in Appendix B. The figure gives an overview of the user interactions providing a good baseline for understanding the available interactions. The interactions aligns with the requirements allowing to manually pre-processing emails as well as training and evaluating a model.

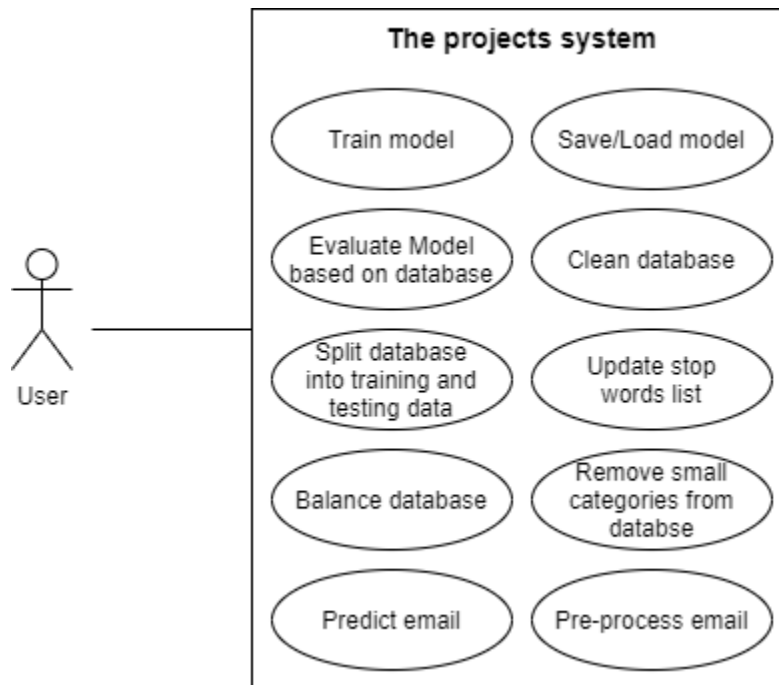


Figure 5.1: Use case diagram

6 | Analysis

Once a high-level understanding of what a system should do is attained through the requirements the analysis workflow can begin. During the analysis workflow models are created to illustrate the essential requirements and characteristics of the system while still focusing on what the system needs to do rather than how[2, p.124]. This chapter will further specify the system core concepts through a domain model, a detailed use case and a sequence diagram with the purpose of laying the foundation of the structure and behavior of the system.

It should be noted that analysis models should be written in the customers language, but for the sake of this report's readability and its intended recipients all models in this chapter will be in English even though IT Relation is a Danish company.

6.1 | Domain Model

A domain model represents the concept of a system and their relations. The systems concepts can be retrieved from the requirements, which states how the system has two main functionalities; it needs to be able to pre-process emails and more importantly classify the emails using a machine learning algorithm. These Concepts are illustrated in the domain model, shown in figure 6.1.

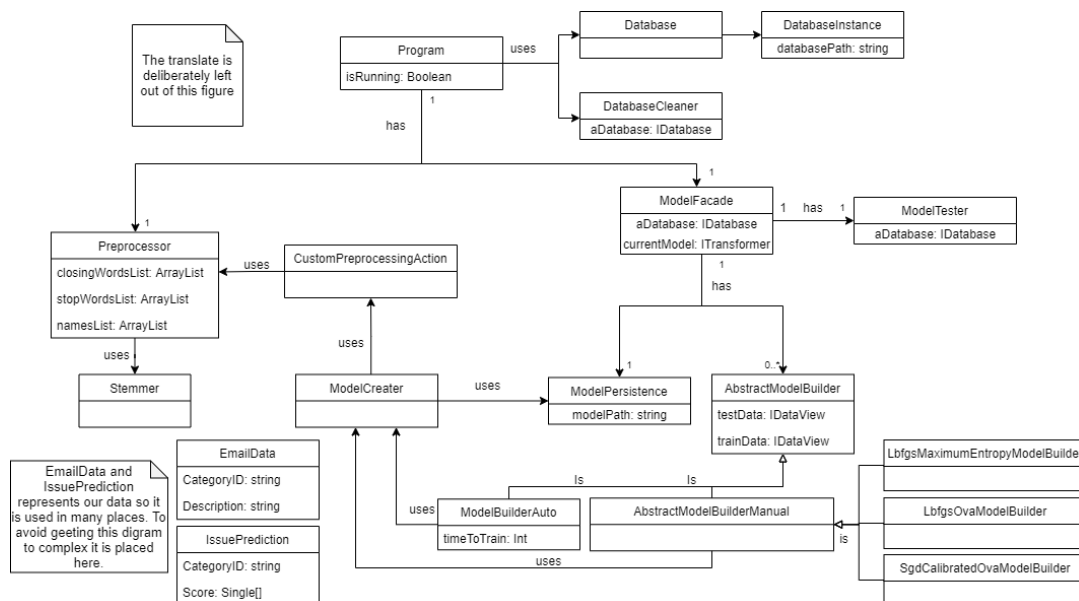


Figure 6.1: Domain model

The domain model is deliberately missing a couple of classes from the system, since they are not needed to understand the systems concept. Firstly translation is missing in its entirety. Since the system is not integrated with IT Relations system the data received from IT Relation was translated once during the project. Because of this the solution does not translate, but only contains the functionality for when the solution will be integrated and needs to be called every time an email is received. Secondly the classes PathHelper and CategoryList are not included in the domain model. PathHelper ensures the correct path is used depending on the computer running the system and CategoryList translates a category id to the categories name for user readability.

6.2 | Detailed Use Case

In this section the use case *Train model* will be described in detail. The *Train model* use case builds a model which make it possible to classify emails into categories. First it pre-processes the database and then it uses the ML.NET framework to train the model meaning it fulfills requirement F002, F003 and F004 and seen in table 5.1.

Detailed Use Case: Train Model
ID: 01
Primary actor: User
Short description: User trains a model based on a given database
Preconditions: System is running
Main flow: <ol style="list-style-type: none"> 1. User types in the 'train' command in the command line. 2. If command is used by typing 'train' followed by a database name. A model is created by: <ol style="list-style-type: none"> 1) Getting training data from the given database 2) Transforming the training data by pre-processing 3) Training the model 4) Evaluating the model 3. Else see alternative flow
Post conditions: A machine learning model is build
Alternative flow: See appendix C

Table 6.1: Detailed Use Case: Train model

6.3 | Sequence Diagram

To give a further understanding of how the classes in the system communicates figure 6.2 shows a sequence diagram of the main flow in the use case *Train model* to illustrate the systems behavior.

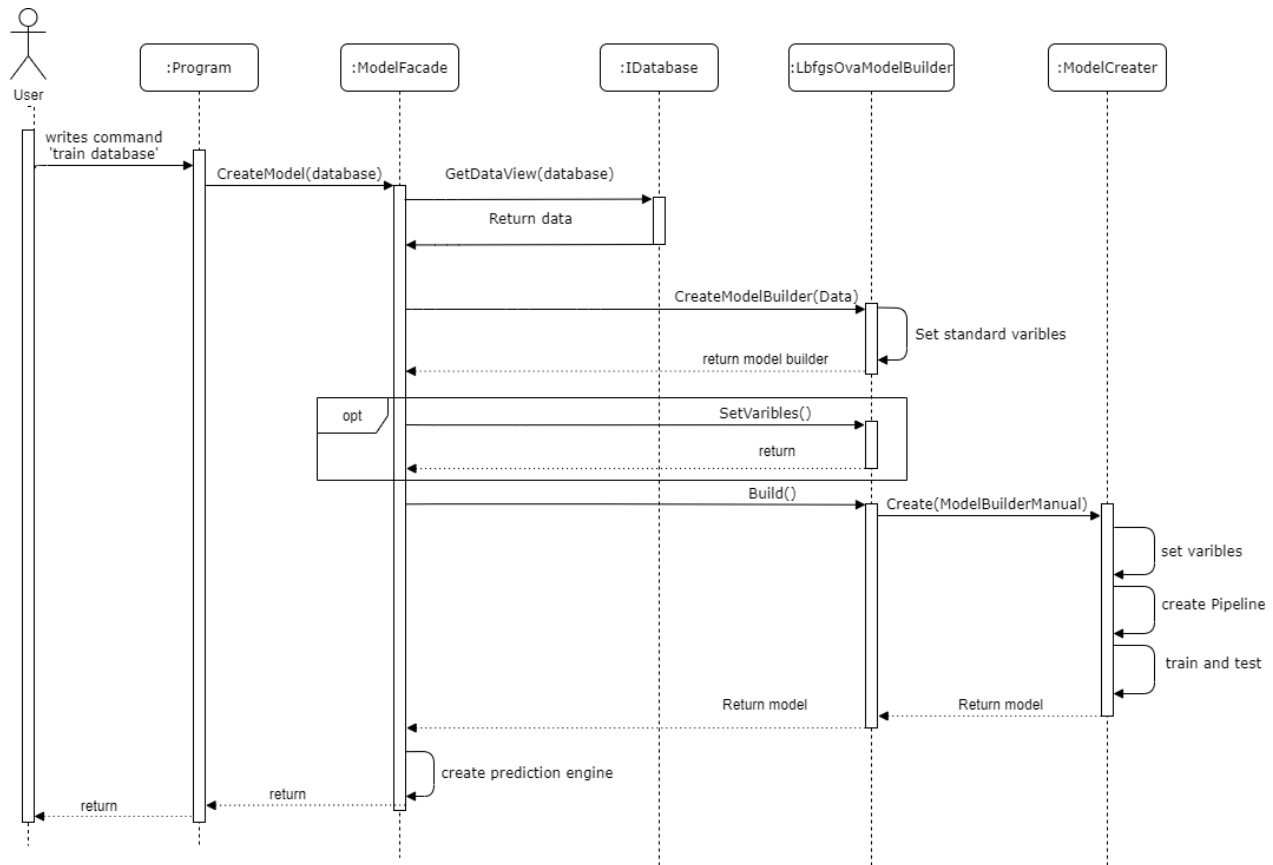


Figure 6.2: Sequence diagram: Train model

7 | Design

Design is a continuation of the analysis workflow in which the focus shifts to determine how functionality will be implemented. Since the project will be handed over to IT Relation and is expected to be used years after delivery, it is especially relevant to keep design and analysis separately. This is because the analysis is needed to understand the "big picture" which will enhance the systems maintainability[2, p. 336].

The chapter will look into the design decisions made for the system. It will explain the system structure, detailing its design patterns as well as describing the external API used.

7.1 | System Structure

The system is divided into 3 main packages; Preprocessing, ML Model and Database, which the **Program** class uses depending on the users interaction, as seen in figure 7.1. An additional package called Translate is also in the system which is used by the pre-processing package.

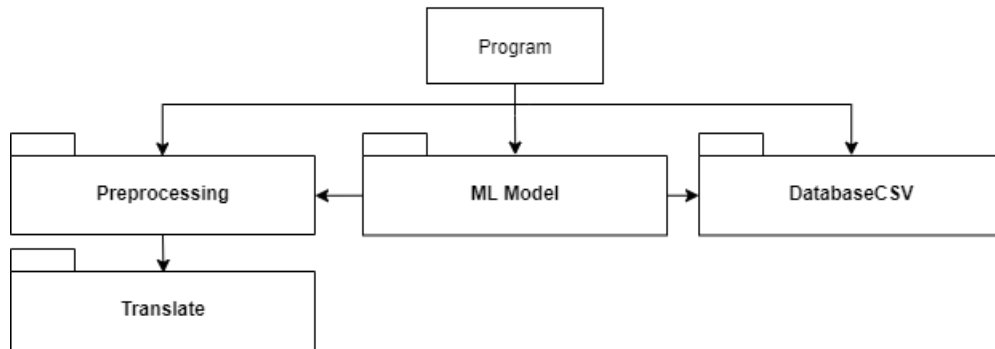


Figure 7.1: Package Diagram

The main method is located in the **Program** class which contains a loop allowing the user to interact with the system. Once the system is started the loop awaits a command. Depending on the command the **Program** will use either the **Preprocessor** from the Preprocessing package to pre-process text, **ModelFacade** from the MLModel package to perform the required action concerning models or interfaces to interact with the Database package to perform actions which require database access. The following subsections will further explain each package.

7.1.1 Preprocessing Package

The Preprocessing package is responsible for the pre-processing of a single text. It not only contains the functionality of the pre-processing, but also all the text files needed to pre-process as well as the snowball stemmer used in the system.

The **Preprocessor** class' *Preprocess* method calls upon multiple private methods to transform the text according to the different steps seen below in figure 7.2.

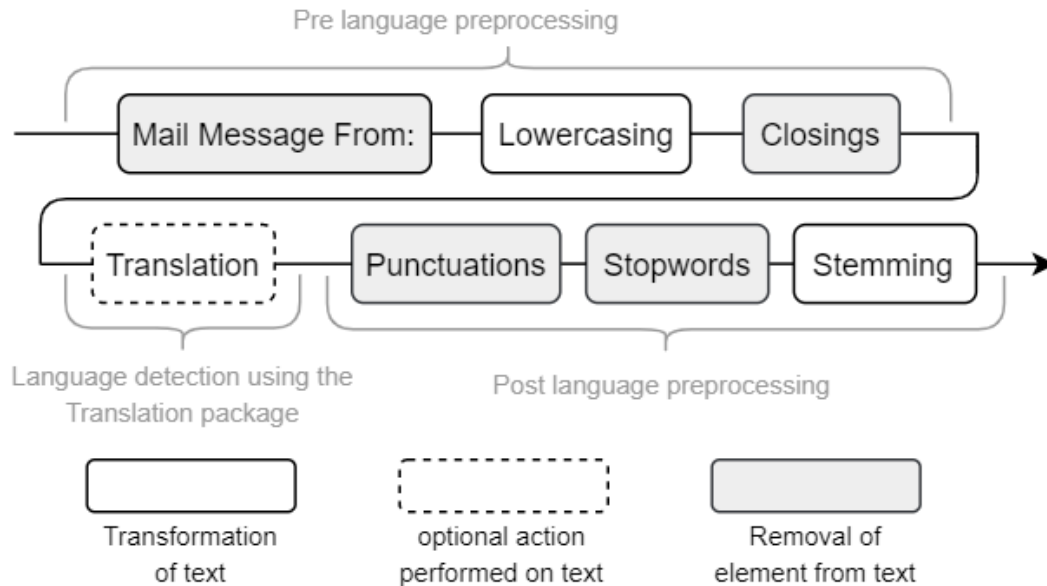


Figure 7.2: The steps taken during the custom pre-processing

The pre-processing steps are further described in section 9.1. The sequence of the steps were chosen with the aim of being as time efficient as possible while also being effective. This is why the removal of "Mail Message From" is the first step as it removes auto generated text the following steps should not take into consideration. The removal of closings happens after transforming to lowercase to be certain no closings are missed because they were written using capital letters. Focusing on time efficiency not only ensures the removal of stop words is faster, since there are fewer words to iterate over but it is also crucial for stemming. Since the stemmer stems every single word to its root, which can be a lengthy process compared to removing stop words it is important it be the last step so as little text as possible had to be stemmed. The use of the **Preprocessor** results in the machine learning model having to iterate over as few words as possible without compensating the emails intent.

7.1.2 DatabaseCSV Package

The DatabaseCSV package is the package responsible for the functionality regarding reading databases, creation of new databases, deleting databases and changing existing ones. At the moment the database works with .csv files but this will change when the system gets integrated with IT Relations system.

To make this as simple as possible to change the database it is implemented using interfaces. This makes it so it is only necessary to change the implementation of those interfaces rather than changing the whole system. The 2 interfaces used for communicating with the database are **IDatabase** and **IDatabaseInstance**. The first interface is implemented by the **Database** class which is used as the entry point. It is what the classes outside or inside the package will use when they communicate with the database. The other interface is implemented by the **DatabaseInstance** class which represents a specific table in the database, which in our case is a .csv file.

Lastly the package has the **DatabaseCleaner** class which uses the above interfaces to transform tables to other tables. This is also done through interfaces to make it simple to change later. As an example the *BalanceData* method creates a new table where each category appears the same number of times.

7.1.3 MLModel Package

The last main package is the MLModel package which contains all classes that interacts with the ML.NET framework and is responsible for the creation and usage of the machine learning models.

This package is the biggest so to make communicating with it as simple as possible it has a Facade class called **ModelFacade**. The package also contains an **AbstractModelBuilder** class and two implementations of that class called **ModelBuilderAuto** and **AbstractModelBuilderManual**. The **AbstractModelBuilderManual** class has three implementations each of which use a different algorithm to create models, the classes are **LbfgsOvaModelBuilder**, **SgdCalibratedOvaModelBuilder** and **LbfgsMaximumEntropyModelBuilder**. Those classes have been created with the builder pattern and are used to build models using the **ModelCreator** class which uses the ML.NET framework to create the machine learning models. Additionally, in the MLModel package there is also a class to save and load the models called **ModelPersistence** and a class for testing/evaluating models called **ModelTester**. There are two classes that hold data, those are **EmailData** which contain the CategoryID and the Description and is what the system uses to represent Emails, and **IssuePrediction** which is the final class a machine learning model will give when it has made

a prediction. It contains the `CategoryID` which is the predicted `CategoryID` and a single-precision floating-point number array `Score` which contains values from 0 - 1 that represents how certain the model is on each `CategoryID`. Lastly the package contains the **PreprocessingCustomAction** class where it is defined how ML.NET should use the **Preprocessing** class in the `Preprocessing` package.

7.1.4 Translate Package

The `translate` package uses the `NTextCat` Nuget to detect language and the Azure cognitive services API to translate. This will be explained further in section 7.4.

7.2 | Design Patterns

Design patterns are reusable solutions for reoccurring problems. This section will go over the different design patterns used in the system and the reasoning for using them.

7.2.1 Facade Pattern

The facade design pattern is a structural pattern that aims to hide the complexity of a system by providing a class that acts as a simple interface that clients can use[24]. The machine learning part of the system is the biggest and most complex of the system, so to make it as simple as possible for IT Realtion to use a Facade class was created. This class is called **ModelFacade**.

7.2.2 Singleton Pattern

A singleton is a creational design pattern that ensures only one instance of a class will be created, while giving global access to that instance[13]. The **Preprocessing** class is a singleton because when the **Preprocessing** class is instantiated it uses .txt files to load stop words and email closings into the system, this can take some time so it is necessary to ensure that it only happens once and not every time something is pre-processed as that would slow down the system substantially.

7.2.3 Builder Pattern

A builder design pattern is used to separate the construction of a complex object with its representation, this allows for a simple step by step creation[23]. Creating a machine learning

model is a complex operation, so to simplify the creation an abstract builder class called **AbstractModelBuilder** was created. It contains 2 different implementation since there are two different ways of building a model.

The first is **ModelBuilderAuto**, where ML.NET have been implemented to build different models using different algorithms and featurization options while only keeping the best, and **AbstractModelBuilderManual**. This class is an abstract class that all the specific algorithm implementation inherit from, it holds attributes and methods they all need. The **ModelCreator** uses a class inherited from the **AbstractModelBuilder** to create a machine learning model using ML.NET.

7.3 | Patterns for Maintainability

As part of the maintainability requirement Better Code Hub's guidelines, as mentioned in subsection 5.3.1, were used to ensure as few code smells were in the system as possible. It is natural when dealing with a deadline to rush programming and produce bad code[16, p. 3], but since messy code and productivity are negatively correlated¹ it is a vicious circle halting the maintainability of a system.

Better Code Hub is a software platform which enables users to check the maintainability of a software system with the help of 10 guidelines[4]. Even though the software it self was not used on the project, guideline 1), 2) and 3), were followed and if violations occurred design patterns were used to resolve them. Guideline 10) was also followed but violations were handled with refactoring rather than patterns. Descriptions of the guidelines can be found in appendix A

To ensure the guidelines were followed there were especially two problems to be mindful about. The first being long methods. Since a big part of the project was researching the best approaches it was also necessary to test multiple different approaches. This lead to classes such as the **Preprocessor** and **DatabaseCleaner** containing large methods hurting the readability of their functionality. To solve this the *Extract function*[7, p. 106] pattern was used. It enabled parts of the functionality to be extracted into its own functions, which lead to shorter units of code, simpler units of code and less duplicated code, but most importantly it made the functions easier to understand and maintain.

The second problem was duplicated code across classes, due to similar functionality needed

¹See glossary for an explanation of 'Negative correlation'

across the system. To accommodate this problem the *Replace Function with Command*[7, p. 337] pattern was used. This design pattern resulted in the classes **PathHelper** and **CategoryList** to be implemented making their functionality accessible for the entire system eliminating the duplicated code.

Patterns are to an extent a flexible and promising way to develop object oriented design, but if overdone can lead to over-engineering[13]. Since they also require knowledge of the pattern, the simple *extract function* and *replace function with command* patterns were chosen to upkeep guideline 1), 2) and 3) without damaging the simplicity or readability of the system.

7.4 | Translation API

Because IT Relations emails come in many different languages it is necessary to translate them all into the same language for a single model to make good predictions on all of them. The Microsoft Azure Cognitive Services API was used for this.

The Cognitive Services API has pre trained machine learning models that clients can access using a web API. The one used in the project is the translator API. IT Relation uses other Microsoft Azure products and even used the same translator for their existing system so it was natural for the project to use it as well. However using the Cognitive services translator costs money so it is necessary to limit the usage of it. This is done by doing part of the pre-processing before translation, removing a majority of text, and by detecting the emails language before calling the service. Additionally, for the project every email was translated in the beginning to avoid doing it every time preprocessing or testing was required.

7.4.1 NTextcat

NTextCat is a language detection NuGet that allows users to detect language. NTextCat is a NuGet where all files are stored locally on the pc, this makes detecting the language faster since it is not necessary to call a web API and wait for a response. It works by having language dictionaries and then comparing a string of text with those dictionaries. Multiple dictionaries are available but the one the project uses is the smallest with only 14 common languages since the languages IT Relation communicates in are within those 14. This is done to increase the accuracy of the language prediction, especially on emails containing very few words since it will have fewer languages to choose between.

7.4.2 Implementation

NTextCat is implemented using the NuGet package manager, which downloads the necessary compiled code and adds a reference to the specific language dictionary the project will use. Since Microsoft Azure Cognitive Services Translator is a web API it is used with Http Requests. The API has a symbol limit so it is necessary to split the long emails up into multiple Http requests which is done by splitting it on every 500 words. Because it costs money to translate it is required to send a subscription key as part of the header so Microsoft knows from which subscription to take the money. Because the project is done for IT Relation their subscription key were used.

8 | Implementation

The aim of the implementation is to produce executable code, by realizing the design decisions [2, p.476]. According to the systems concepts mentioned in section 6.1 the implementation has two main objectives; implement code which can pre-process a string of text and classify it by its category, with the classification being most important. To obtain these objectives the approach shown in table 8.1 was used.

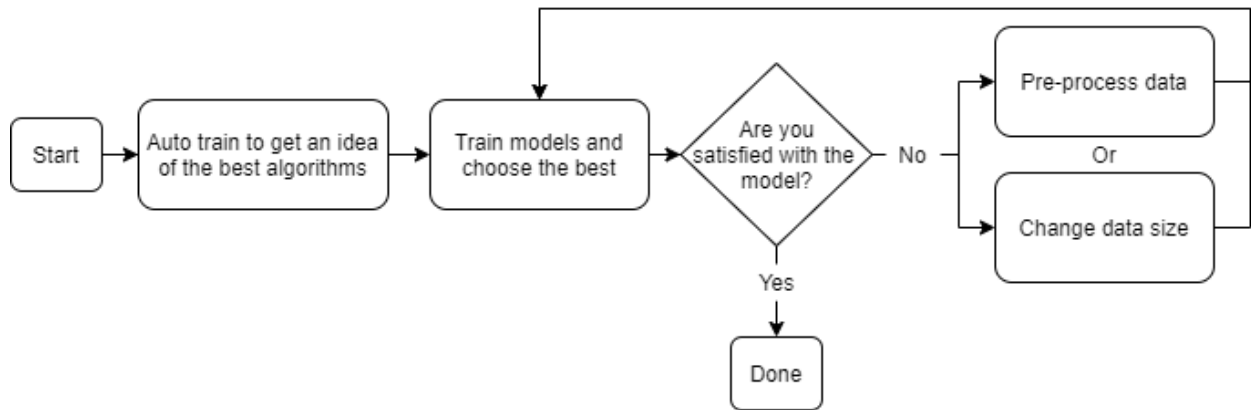


Figure 8.1: The overall approach to creating the systems model

As seen in figure 8.1 The implementation began with using ML.NETs option to auto train, which trains multiple different models to get an idea of what models to work with. Then continue training on chosen models. Since the first chosen model did not produce a satisfactory accuracy, the data was pre-processed and then trained again. Lastly to further enhance the model balancing the data was tested and models were trained one last time. As the figures shows the entire process was an iterative process, since every modification in the data made it necessary to retrain models. This chapter will first explain how a database is accessed and afterwards the implementation of the figures steps will be explained. The results from testing the different models and the steps taken to enhance them will however be explored in chapter 9.

8.1 | Implementation of Database Access

Since the system is build around using .csv files as databases the system contains methods to read and create databases. To ensure a loose coupling between the classes who

access the database the interfaces **IDatabase** and **IDatabaseInstance** were used. The **Database** class implements the **IDatabase** interface and is the main point of entry to the Database Package. All its methods returns a **DatabaseInstance** class which implements the **IDatabaseInstance** interface.

```
1 public static void Main(string[] args)
2 {
3     IDatabase aDatabase = new Database();
4     ModelFacade mf = new ModelFacade(aDatabase);
5     ...
6 }
```

Listing 1: The Systems main method instantiates a **Database** class and parse it to the **ModelFacade**

```
1 public void ManualCreateModel(string trainingDatabase)
2 {
3     IDatabaseInstance databaseGetter = aDatabase.GetDatabase(trainingDatabase);
4     IDataView trainingData = databaseGetter.GetDataView(context);
5     ...
6 }
```

Listing 2: The **ModelFacade** uses the **IDatabase** to access the database .csv files

8.2 | Auto Training with ML.NET

In ML.NET a trainer is a component executing an algorithm on a specific task. Different algorithms can be applied to different tasks as long as that algorithm has an implemented trainer. This project was limited to the amount of usable algorithms as it was only possible to use algorithms that had a multiclass classification trainer or a binary classification trainer implementation in ML.NET. In the beginning of the project research was conducted to get an idea of what trainers would be relevant and then during implementation the **ModelBuilder-Auto** was implemented to use ML.NETs option to auto train models. The auto option the framework offers trains multiple different trainers and tries different customization options to produce a model the results can be seen in table 8.1.

Trainer	Process	Algorithm	Task	Accuracy
LbfgsLogisticRegressionOvA	Logistic regression	L-BFGS	Binary	65.19 %
SgdCalibratedOvA	Logistic regression	SGD	Binary	64.99 %
LbfgsMaximumEntropy	Logistic regression	L-BFGS	Multiclass	63.12 %

Table 8.1: Trainers used in the system

From the auto test the LbfgsLogisticRegressionOvA trainer was chosen to continue working on, since it had the highest accuracy. How trainers executing a binary classification task was used on a multiclass classification problem will be explained in section 8.3.2.

8.2.1 Implementation of Auto Training

As mentioned in section 7.1.3 the **ModelFacade** is used to simplify the communication between the user and the ML.NET Framework. Because of this the *Train model* command, which can be used to auto train models, calls on the **ModelFacade** which creates a new instance of **ModelBuilderAuto** with a ML context and the given training set and sets the time to train as seen in listing 3.

```

1 public void AutoCreateModel(string trainingDatabase, UInt32 timeToTrain)
2 {
3     IDatabaseInstance databaseGetter = aDatabase.GetDatabase(trainingDatabase);
4     IDataView trainingData = databaseGetter.GetDataView(context);
5
6     ModelBuilderAuto mb = new ModelBuilderAuto(context, trainingData);
7
8     mb.timeToTrain = timeToTrain;
9     currentModel = mb.Build();
10
11     CreateNewPredictionEngine();
12 }

```

Listing 3: The *AutoCreateModel* method in **ModelFacade**

The **ModelBuilderAuto** builds an auto model by using the class' *Build* method which uses the **ModelCreator** class' *Create* method. The *Create* method creates an IEstimator pipeline which adds a new column with the pre-processed text from the emails and removes the column with the original text. It will then run an experiment testing the different trainers and return the best model. The *Create* method is seen in listing 4.

```

1 public static ITransformer Create(ModelBuilderAuto mb)
2 {
3     var pipeline = mb.context.Transforms.CustomMapping(
4         new PreprocessingCustomAction().GetMapping(), contractName: "Preprocessing")
5         .Append(mb.context.Transforms.DropColumns("Description"));
6
7     //Transforming data
8     var textTransformer = pipeline.Fit(mb.trainData);
9     IDataView transformedData = textTransformer.Transform(mb.trainData);
10
11     MulticlassExperimentSettings experimentSettings = SetupExperiment(mb);
12
13     //Creates the experiment with the created settings
14     var experiment = mb.context.Auto().CreateMulticlassClassificationExperiment(
15         experimentSettings);
16
17     ExperimentResult<MulticlassClassificationMetrics> experimentResult = RunExperiment(
18         mb, transformedData, experiment, textTransformer);
19
20     var bestRun = experimentResult.BestRun;
21     PrintBestRun(bestRun);
22
23     //Return the best model, utilising the fact that transformers can be linked so that
24     //we here only return 1 transformer but it preprocess the data and makes predictions
25     return textTransformer.Append(bestRun.Model);
26 }

```

Listing 4: The *Create* method creating a auto model in **ModelCreator**

Once the model is trained and tested the "currentModel" variable is updated and the prediction engine is updated with the *CreateNewPredictionEngine* method, as shown in listing 3. The prediction engine is the class ML.NET uses to make single predictions. It uses three elements:

- A class, in this system **EmailData**, representing the data before any pre-processing or prediction.
- A class, in this system **IssuePrediction**, representing the output after both pre-processing and the model making a prediction.
- The current model as an argument to transform the data and make a prediction.

The prediction engine is created as shown in listing 5.

```
1 private void CreateNewPredictionEngine()  
2 {  
3     predEngine = context.Model.CreatePredictionEngine<EmailData, IssuePrediction>(  
4         currentModel);  
5 }
```

Listing 5: The *CreateNewPredictionEngine* method in **ModelFacade**

Once a prediction engine have been created prediction can be made by using the **ModelFacade** class' *PredictIssue* method which takes a string as an argument. The method will then use the prediction engine to pre-process and predict the category the string belongs to. Because the prediction engine expects to get the class **EmailData** the string has to first be transformed into an **EmailData** object.

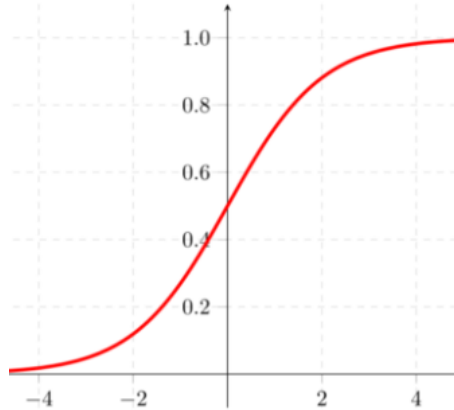
8.3 | Machine Learning Trainers

The three trainers obtained from the auto training, LbfgsLogisticRegressionOvA, LbfgsMaximumEntropy and SgdcalibratedOvA all have unique combinations of algorithms and tasks even though they all use Logistic regression. Before continuing to explain the implementation of the system, Logistic regression and how binary classifiers were used for a multiclass classification task will be explained. Furthermore explanations of the L-BFGS and SGD algorithms can be found in appendix D.

8.3.1 Logistic Regression

Logistic regression gets it name from the function it is based on, the logistic function, also called the Sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}}$$



Source: [26]

Figure 8.2: The logistic function also called Sigmoid

The output of this function is always between the values 0 - 1. Just as linear regression tries to fit a linear function to the data, as explained in chapter 2, logistic regression tries to fit a logistic function to the data. Since the output of the logistic function is between 0 - 1 it can be used for classification by treating the output y as the probability of some x input being part of a class $y = 1$ or $y = 0$. If the probability is higher than 0.5 it is defined as being part of the class 1 if it is below 0.5 it defined as being part of the class 0. Logistic regression as linear regression uses an equation where the inputs x combines linearly using weights w .

$$f(x) = \frac{1}{1 + e^{-(w_0 + x * w_1)}} \quad \text{or with n features} \quad f(x) = \frac{1}{1 + e^{-(w_0 + x_1 * w_1 + x_2 * w_2 \dots)}}$$

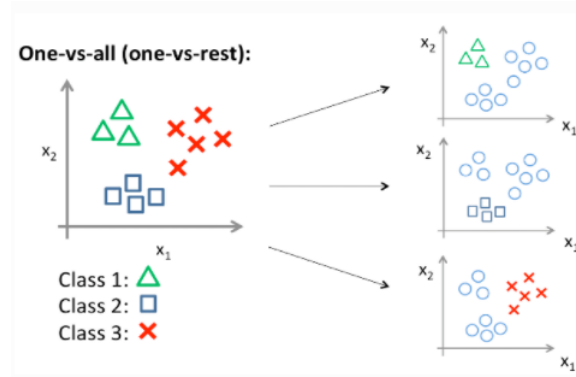
Like linear regression a dummy feature for $x_0 = 1$ can be added to write the function[28]:

$$f(x) = \sum_{i=0}^n \frac{1}{1 + e^{-(w_i * x_i)}}$$

This function is the main difference between logistic regression and linear regression, since it maps the output to be between 0 - 1. As with linear regression the goal is to find the values for the weight that make the best predictions.

8.3.2 Binary Classification to Multiclass Classification

ML.NET implements a One vs All (OvA) multiclass classification trainer, sometimes called one vs rest. A OvA trainer modifies a binary classification trainer to be used as a multiclass classification trainer. It does this by creating a binary classification trainer for each class that predicts whether data belongs to the class or not.



Source: [19]

Note: The circles represents the rest of the classes

Figure 8.3: How One-versus-All makes binary classification into multiclass classification

Because of this every binary classification trainer that ML.NET implements can be used as a multiclass classification trainer, which is how both the `LbfgsLogisticRegressionOvA` trainer and `SgdcalibratedOvA` trainer was used in this project. The implementation of this is shown in listing 6.

```

1 public override EstimatorChain<KeyToValueMappingTransformer> AddTrainerToPipeline()
2 {
3     return context.MulticlassClassification.Trainers.OneVersusAll(
4         context.BinaryClassification.Trainers.LbfgsLogisticRegression(
5             l1Regularization: l1Regularization,
6             l2Regularization: l2Regularization,
7             optimizationTolerance: optimizationTolerance,
8             historySize: historySize,
9             enforceNonNegativity: enforceNonNegativity)
10     )
11     .Append(context.Transforms.Conversion.MapKeyToValue("PredictedLabel"));
12 }

```

Listing 6: The *AddTrainerToPipeline* method in `LbfgsOvaModelBuilder`

8.4 | Model Training with ML.NET

As mentioned the system has two options to train models, it can create a manual or an auto model. The manual model creates and tests a model with a given trainer. The two options are very similar in implementation with the few differences being explained in this section.

As with the auto training it is the *Train model* command which starts the creation of the manual model by calling on the **ModelFacade**. The difference between the *AutoCreateModel* method and the *ManualCreateModel* method is that an implementation of the **Abstract-ModelBuilderManual** class, such as the **LbfgsOvaModelBuilder**, is instantiated instead of a **ModelBuilderAuto** this can be seen in listing 7.

```
1 LbfgsOvaModelBuilder mb = new LbfgsOvaModelBuilder(context, trainingData);
```

Listing 7: Part of the *ManualCreateModel* method in **ModelFacade**

Just as with the auto training a model is build using the **ModelCreator** class. The *Build* method for a manual model creates a pipeline which first transforms the CategoryID into an integer key which ML.NET prefers to use when dealing with classification, this key is called the Label¹. The custom action which performs this systems pre-processing is then added to the pipeline, and then the pre-processed text is featurized using ML.NET. Featurizing will be described further in subsection 9.2.1. The output of this custom action is a Features column².

Then the unnecessary columns are removed from the data, the machine learning trainer is added and lastly the key integers are transformed back into readable category IDs. The *Create* method is seen in listing 8.

¹In ML.NET the label is the element to predict, in this project the emails category id

²In ML.NET the features column is the input trainers use to make predictions, in this project the emails description

```

1 public static ITransformer Create(AbstractModelBuilderManual mb)
2 {
3     var options = mb.GetTextFeaturizingOptions();
4
5     var modelPipeline = mb.context.Transforms.Conversion.MapValueToKey(inputColumnName:
6         "CategoryID", outputColumnName: "Label")
7         .Append(mb.context.Transforms.CustomMapping(new PreprocessingCustomAction().
8             GetMapping(), contractName: "Preprocessing"))
9         .Append(mb.context.Transforms.Text.FeaturizeText("Features", options,
10             "PreProcessedText"))
11         .AppendCacheCheckpoint(mb.context)
12         .Append(mb.context.Transforms.DropColumns("Description", "PreProcessedText",
13             "CategoryID"))
14         .Append(mb.AddTrainerToPipeline());
15
16     SetTestData(mb, out IDataView trainingData, out IDataView testData);
17
18     Console.WriteLine($"Training, testing and building: {mb.modelName}");
19     var trainStopWatch = new Stopwatch();
20     trainStopWatch.Start();
21     var trainedModel = modelPipeline.Fit(trainingData);
22     trainStopWatch.Stop();
23
24     EvaluateModel(mb, testData, trainedModel, trainStopWatch);
25     return trainedModel;
26 }

```

Listing 8: The *Create* method creating a manual model method in **ModelCreator**

After the pipeline is created the test and training data is set, if no test data was given the training data is split into a 80% training set and a 20% testing set. The pipeline is then fitted on the training data meaning the pipeline will go from defining what to do to how to do it. Part of this process is the trainer learning how to classify which means it will be trained and become a model. When the training is done the model is evaluated using the testing set.

8.5 | Implementation of Custom Pre-processing

Since ML.NET has an option to use a custom action the pre-processing was integrated with the framework by implementing it as a custom action. This required two classes the **Pre-**

processingCustomAction class which defines what the custom action should do using the *CustomAction* and *GetMapping* methods which uses the *PreProcess* method in the **Preprocessor** class. The second class is the **CustomMappingOutputPreprocess** which defines how the data looks after the custom action. It was also necessary to add the *CustomMappingFactoryAttribute("Preprocessing")* attribute to the class since this attribute is how ML.NET finds the **Preprocessor** class when the model is loaded, if this was not here it would not be possible to save or load models that use the custom action. The implementation of the **PreprocessingCustomAction** class can be seen in listing 9. And how it is applied to the pipeline can be seen in listing 8 on line 7 and 8.

```

1  [CustomMappingFactoryAttribute("Preprocessing")]
2  class PreprocessingCustomAction : CustomMappingFactory<EmailData,
3      CustomMappingOutputPreprocess>
4  {
5      private static Preprocessor preprocessor = Preprocessor.GetInstance();
6
7      public void CustomAction(EmailData input, CustomMappingOutputPreprocess output)
8      {
9          output.PreProcessedText = preprocessor.PreProcess(input.Description);
10     }
11
12     public override Action<EmailData, CustomMappingOutputPreprocess> GetMapping()
13         => CustomAction;
14 }
15
16 public class CustomMappingOutputPreprocess
17 {
18     public string PreProcessedText { get; set; }
19 }

```

Listing 9: The **PreprocessingCustomAction** class

8.6 | Implementation of ML.NET Pre-processing

Besides the custom pre-processing ML.NET also offers its own options to prepare data, which will be explained more in depth in section 9.2. The definition of these ML.NET pre-processing options are on the **AbstractModelBuilderManual** class in the *GetTextFeaturizingOptions* method as seen in listing 10.

```
1  public TextFeaturizingEstimator.Options GetTextFeaturizingOptions()
2      {
3      return new TextFeaturizingEstimator.Options()
4      {
5          CharFeatureExtractor = charFeatureExtractor,
6          Norm = norm,
7          WordFeatureExtractor = wordFeatureExtractor,
8
9          //This is all something we do in our preprocessing
10         //so to save time we stop it from happening here
11         CaseMode = TextNormalizingEstimator.CaseMode.None,
12         KeepNumbers = true,
13         KeepPunctuations = true,
14         KeepDiacritics = true,
15         StopWordsRemoverOptions = null
16     };
17 }
```

Listing 10: The *GetTextFeaturizingOptions* method in **AbstractModelBuilderManual**

Many of the options in the text featurizer, listing 10, are set to the value *true* meaning they will be ignored by the featurizer. This is done since many of those options are already done in our custom pre-processing step with more control. The other options are defined using the values in the **AbstractModelBuilderManual** object. Lastly when the pipeline is created the *FeaturizeText* is appended with the chosen options as seen on line 9 in listing 8.

9 | Experimental Validation

An important aspect of this project was to constantly test the approaches being taken, to ensure every step of the pre-processing was beneficial and resulted in the best possible model. This chapter will discuss the results from testing the different steps taken to enhance the model, such as the pre-processing and balancing of the data.

9.1 | Testing of Custom Pre-processing

As mentioned in section 8.4 the system uses custom pre-processing, which was not only a requirement gathered from the meeting with IT Relation, as mentioned in chapter 5, but also used to enhance the accuracy of the machine learning model. This section will explain and discuss the different approaches. Through out the section results from multiple test will be displayed all of which were gathered using the LbfgsLogisticRegressionOvA trainer on a data set only containing 3300 emails in each of the 35 categories. The data set was split into a train set containing 80% of the emails and a test set which contained the other 20%. It was chosen to test with the 35 categories since it was the amount IT Relations existing system had used which makes comparing the models easier. The smallest of these 35 categories contained roughly 3300 emails and it was therefore decided to balance the data to 3300 emails since a smaller data set would be more time efficient.

To give a better overview of the different approaches taken to enhance the custom pre-processing figure 9.1 illustrates the process.

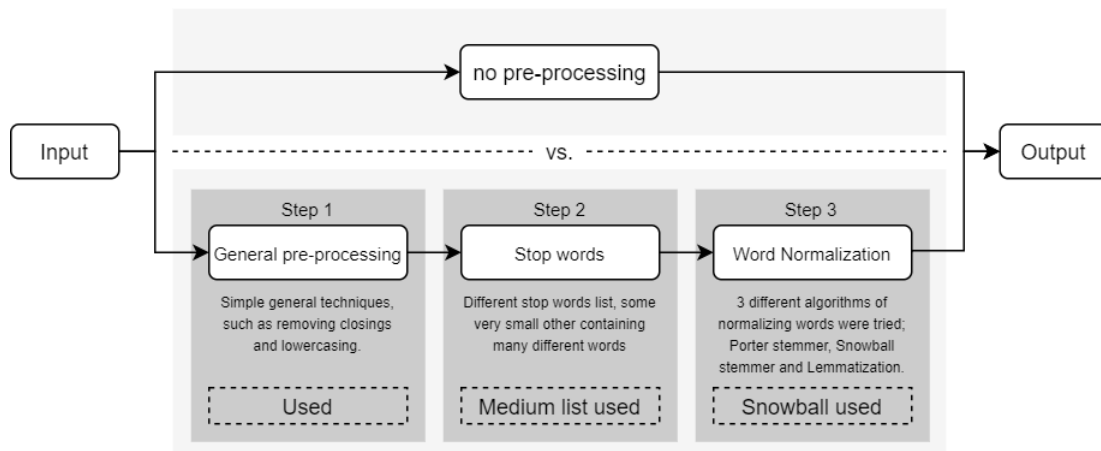


Figure 9.1: The three steps taken during the custom pre-processing

As figure 9.1 shows both using pre-processing and not was tried, to see how it would affect the accuracy of the machine learning model. The final evaluation of using pre-processing will be explained in section 9.3, the rest of this section will instead detail the implementation of the three steps taken during pre-processing.

9.1.1 Step 1) General Pre-processing

As explained in section 7.1.1 the pre-processing contains multiple different methods to transform the text. The general pre-processing consist of the following pre-processing steps; lower-casing, removal of closings, removal of "Mail Message From", removal of punctuation marks and translating the text, which are all also done in the existing system as explained in section 3. This means the general pre-processing is everything except the removal of stop words and word normalization. Due to the simplicity of the general pre-processing methods they will not be further explained, but are described in detail in appendix E. Other methods such as removal of symbols, numbers, diacritics and apostrophes, which are not in the existing system, was also tested. They ended up worsening the models accuracy and is therefore not part of the general pre-processing. The results from testing the four methods can however be seen in appendix F and the results are believed to be due to over pre-processing meaning to much information is being removed.

The effects of the general preprocessing was tested and can be seen in table 9.1.

Option	Accuracy	Training time	Pre-process time
Used	63.97 %	142,644 ms	2,661 ms
Not used	62.04 %	254,247 ms	- ms

Note: Training time is both pre-processing time and time to train the model.

Table 9.1: General pre-processing results

The test showed that the general pre-processing do speed up the process of training the model making it almost twice as fast. Furthermore it enhances the accuracy by 1.93%. This proves the data does contain noise which harms the machine learning models accuracy and especially the training time. Due to the test results supporting the use of the general pre-processing it was decided to use in the system.

9.1.2 Step 2) Stop Words

Stop words are the most common words in a language and therefore can be filtered out, since they do not provide any meaning to a text. The removal of stop words should decrease the training time. To combat the fact that a universal stop words list does not exist, multiple different stop word lists were tested. To make it simple to modify the lists the system uses .txt files, which the **Preprocessor** loads and uses. The results from the test can be seen in table 9.2.

List name	Word count	Accuracy	Training time	Pre-process time
Mega	797	64.48 %	135,603 ms	32,829 ms
Large	225	64.53 %	119,387 ms	14,197 ms
Medium	127	64.63 %	117,566 ms	8,479 ms
Small	32	64.26 %	121,563 ms	5,023 ms
No list	0	63.97%	132,634 ms	3,658 ms

Note: Training time is both pre-processing time and time to train the model.

Table 9.2: Word List Results

The results showed how even though the word lists add additional pre-process time compared to not using a list it does not negatively affect the training time since there are less words for the model to learn. Pre-processing time increases with the list size because the system needs to look through more words which takes time, even if time is saved in later pre-processing steps the added time to look through the list increases the overall time too much. However when it comes to training time there is sweet spot where time is saved because the amount of features an algorithm have to look at are decreased without the pre-processing time being increased too much. The medium sized list had both the best accuracy and the best training time and it is therefore used in the project.

9.1.3 Step 3) Word Normalization

Word normalization is the process of changing words to their root or dictionary form. There are two overall approaches to normalize words; stemming and lemmatization. Both were tested during this project since they have the same aim of mapping similar words to the same form which should enhance the accuracy of a machine learning model.

Stemming is the process of reducing words down to their base form called the stem. This makes it so words like "Jump", "Jumping" and "Jumped" all gets reduced to the same stem "Jump". Similarly lemmatization reduces words to a base form that is still a valid word,

which is called lemma. Unlike stemming, lemmatization requires identifying the intended part of speech and meaning of a word in a sentence to correctly map it to a lemma.

Early in the project it was hypothesised that stemming would be the superior choice due to it being a faster method and the machine learning model not needing an actual word such as the lemmatization method produces. Because of this it was decided to test two stemmers, Porter and Snowball as well as a single implementation of lemmatization. Porter and Snowball was chosen as the implementation of stemmers since research found them to be the most commonly used[8]. All three implementations were tested along side with combinations of the two stemmers to see if it would enhance the result. The results from the test can be seen in table 9.3

Method	Average accuracy	Training time	Pre-process time
Snowball	64.65 %	126,774 ms	19,634 ms
Porter	64.44 %	144,106 ms	15,408 ms
Snowball-Porter	64.46 %	141,912 ms	25,819 ms
Porter-Snowball	64.45 %	133,533 ms	27,526 ms
Lemmatization	64.64 %	675,021 ms	73,623 ms
No word normalizer	64.63 %	132,634 ms	7,972 ms

Note: Training time is both pre-processing time and time to train the model.

Table 9.3: Word normalization results

According to the test both stemming and lemmatization will improve the machine learning models accuracy by a small amount, but due to the increase in both the training and pre-processing time lemmatization was not considered as an viable option for the system.

The snowball stemmer performed slightly better than the Porter stemmer. In stemming there are only two types of errors over stemming, where words that do not share a meaning gets reduced to the same stem. And under stemming, where words with the same meaning do not get reduced to the same stem. After comparing stemmed emails, it became clear the Porter stemmer was prone to under stemming. This can also explain why even though the Snowball stemmers pre-processing time is worse than porters the overall training time is still better, since the output of the Snowball stemmer stems more words to the same root leaving less features for the machine learning model to consider. Using both stemmers also did not produce a better accuracy than with just the Snowball stemmer and was therefore not considered as an option. Because of the test results it was decided to use the Snowball stemmer in the system.

9.2 | Testing of ML.NET Pre-processing

The following subsections will detail the ML.NET pre-processing the system does to further enhance the accuracy. As with the custom pre-processing all tests in the section were done using the LbfgsLogisticRegressionOvA trainer on 35 categories with 3300 emails in each.

9.2.1 Featurize Text

Featurize text is a text transformation in ML.NET which transforms text to numbers ensuring machine learning algorithms can understand them. This is the last step in pre-processing meaning the output of the featurize text transformation is the input for the machine learning algorithm. Featurize text is done in 4 steps which will be explained in more details in the following sections:

1. Create all possible bag of word N-grams and bag of char N-grams from the corpus, these are called "Features".
2. Count how many times each feature appears and the specific email but also how many emails contain each feature.
3. Apply TF-IDF using the above numbers. This transforms the email into a key:value pair where a Key is a feature and the value is the TF-IDF score.
4. Use a normalization function to transform every Feature score to a value between 0 and 1.

9.2.2 Bag of Word N-Grams

The first of ML.NETs methods used is the option to transform text into bag of word N-grams. Bag of words is a natural language processing technique that represents how many times a word appear in a text. N-grams are sequences of n words. So bag of word N-Grams allows the algorithm to look at N sized sequences of words to not only look at single words but also the context they are used in. As a standard, ML.NET uses unigrams¹ however this can be changed with ML.NETs "WordBagEstimator.Options" during the implementation to allow for testing of different sizes of N.

UseAllLengths is another feature that ML.NET has, if it is used it will create N-Grams of all sizes up to N. So if trigrams UseAllLengths is used the system would create a bag of

¹See glossary for an explanation of 'Unigram'

unigrams, bigrams and trigrams. This will create many features which most likely will slow down the algorithm but also gives it more information which could make it better.

Word NGramSize	Accuracy	Training Time
Unigrams	64.64 %	124,562 ms
Bigrams	63.77 %	266,898 ms
Bigrams, UseAllLengths	64.40 %	316,368 ms
Trigrams	63.12 %	377,546 ms
Trigrams, UseAllLengths	63.93 %	634,602 ms

Table 9.4: Results from testing the use of n-grams

Unigrams ended up being both the fastest and most accurate. This means that in this data set the order of which words appear in are not as important as just which words appear. Time wise it makes sense that unigrams are the fastest as it is the one with the least features. Because unigrams are both the fastest and the most accurate it is what this project will be using.

9.2.3 Bag of Char N-Grams

Bag of Char N-Grams or, Bag of Character N-Grams, works similarly to Bag of word N-Grams, but whereas Word N-Grams split a sentence up into sequences of N words, Char N-Grams split the sentence up into sequences of N characters. As a standard ML.NET uses Char-trigrams but as with word N-grams it is possible to modify this which was also tested during the implementation.

Char NGramSize	Accuracy	Training Time
Turned off	61.86 %	63,956 ms
Unigrams	62.46 %	93,599 ms
Bigrams	63.81 %	101,067 ms
Bigrams, UseAllLengths	63.08 %	118,461 ms
Trigrams	64.64 %	124,167 ms
Trigrams, UseAllLengths	63.26 %	174,020 ms
4-grams	64.50 %	210,500 ms
4-grams, UseAllLengths	63.27 %	370,564 ms
5-grams	64.17 %	376,385 ms
5-grams, UseAllLengths	63.27 %	854,247 ms

Table 9.5: Results from testing the use of character n-grams

The test showed the ML.NET default of Char-trigrams was indeed the option with the highest accuracy and is therefore used in the system. Not using it at all was the fastest option but the increase in accuracy by using char-trigrams was deemed worth the increase in time.

9.2.4 TF-IDF

TF-IDF is a method to estimate how important a word is to a document, or in this case a n-gram to an email, compared to the corpus of emails. The following equation is used to calculate TF-IDF for each n-gram:

$$IDF = \log \frac{\text{number of emails in the corpus}}{\text{number of emails that contain the } n - \text{gram}}$$

The result is the inverse number of how many emails contains the n-gram, meaning the more emails that contain a n-gram the lower the score. This result is then multiplied with the term frequency, which is the number of times the n-gram appears in a specific email. This is done for all words and the higher the score the more abundant and important the word is deemed to a specific email. By default ML.NET does not use TF-IDF.

TF-IDF can be used in ML.NET with both bag of word n-grams and bag of char n-grams. So both were tested and the results are as follows.

TF-IDF	Accuracy	Training Time
None	64.64 %	125,900 ms
Words	65.18 %	125,122 ms
Chars	64.91 %	126,002 ms
Words and Chars	65.46 %	124,214 ms

Table 9.6: Results from testing TF-IDF bag of words

For both bag of word n-grams and bag of char n-grams there is a small increase in accuracy. When using both there is a 0.82% accuracy increase compared to using neither and TF-IDF was therefore included in the system.

9.2.5 Normalization

Normalization is the process of adjusting the feature's TF-IDF score gained from bag of words n-grams and bag of char n-grams to a number between 0 - 1. ML.NET offers 3 different normalisation all as well as none with L2 being the default function but all were tested.

Normalization function	Accuracy	Training Time
None	64.42 %	1,022,357 ms
L1	57.15 %	106,917 ms
L2	65.46 %	123,493 ms
Infinity	67.81 %	149,957 ms

Table 9.7: Word List Results

From the test results it can be seen how the infinity function is the most accurate and is what is used in this system. The infinity normalisation functions is a function that divide the highest feature score with every feature score. This makes it so the highest feature score will have the value 1 and every other feature score will be equal to or less than 1.

9.3 | Evaluation of Pre-processing

With the results from all the minor tests of the different steps in the custom pre-processing as well as the ML.NET pre-processing the entire use of pre-processing can be evaluated based

on the results from the test shown below in table 9.8. The test were done using the LbfgsLogisticRegressionOvA trainer on the same data set with 35 categories and 3300 entries in each as used during the minor tests.

Option	Accuracy	Training time
Pre-processing used	67.81 %	149,957 ms
Not used	62.04 %	254,247 ms

Table 9.8: Results from testing the entire pre-processing on the test data set

From the test results it can be deduced that the pre-processing does enhance the LbfgsLogisticRegressionOvA based model. The model with the pre-processing had a 5.77 % accuracy increase and a 104,290ms decrease in training time. Because of the results from the test it was decided to use both the custom-pre-processing steps and the ML.NET pre-processing in the system. Figure 9.2 shown below illustrates the final steps used in the systems pre-processing.

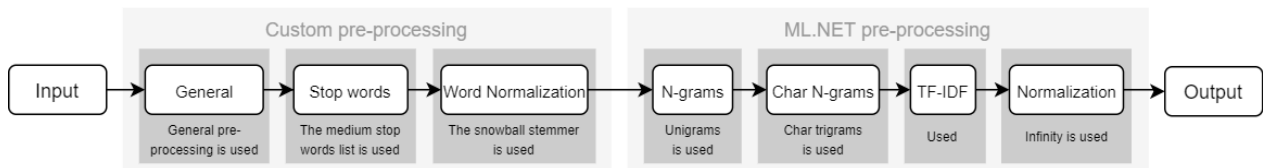


Figure 9.2: Final steps of the systems pre-processing

9.4 | Testing of Database Modification

The last approach taken to improve the accuracy of the model has to do with changing the size of the data. There were two hypotheses behind this approach, firstly the size of the data may impact the different models and the previous test of them may no longer accurately represent their accuracy. Secondly, since the amount of data in each category varies drastically the accuracy could be affected by making the models biased towards the categories with more data, since there would not be enough data in smaller categories to support a prediction. To combat these problems the method *BalanceData* was implemented in the **DatabaseCleaner** class with the functionality of modifying a data set to only contain categories with a specific amount of emails. This method was how the data set used in previous tests was made.

In order to test the first hypothesis, which assumes a balanced data set may change what

trainer is the correct to use, different trainers were tested once again. The test data set previously used with 35 categories and 3300 emails was used as the balanced data. The results from testing the different trainers on balanced data is shown below in table 9.9. Micro accuracy is the overall accuracy for the entire data set², where as macro accuracy is calculated by calculating the accuracy of each category and then averaging them.

Trainer	Micro accuracy	Macro accuracy	Training time
LbfgsLogisticRegressionOvA	67.81 %	67.92 %	149,957 ms
SgdCalibratedOvA	68.36 %	68.21 %	152,165 ms
LbfgsMaximumEntropy	66.21 %	66.23 %	357,001 ms

Table 9.9: Test results from training and testing multiple trainers with a balanced data set

The test proved the first hypothesis can be confirmed since the test shows how the SgdCalibratedOvA is the most accurate when using a balanced size of entries with the pre-processing. However since IT Relations do not receive a balanced amount of emails it is therefore also necessary to test the balanced models on unbalanced data.

As mentioned the amount of data in IT Relation's database varies drastically between categories with the smallest category containing 3,315 emails and the largest containing 70,114 emails. Since it is assumed that IT Relation will continue to receive emails in that ratio, it was used to create an unbalanced test data set The two best models trained on the balanced set seen in table 9.9 were therefore tested on this unbalanced set alongside with models created using the same trainers but on the entire data³, as seen in table 9.10.

Trainer	Trained on	Micro A.	Macro A.	Training time
LbfgsLogisticRegressionOvA	Balanced	54.90 %	60.33 %	149,957 ms
LbfgsLogisticRegressionOvA	Unbalanced	68.57 %	62.53 %	1,823,789 ms
SgdCalibratedOvA	Balanced	55.66 %	61.64 %	152,165 ms
SgdCalibratedOvA	Unbalanced	68.34 %	62.29 %	2,526,471 ms

Note: A. = Accuracy

Table 9.10: Results from testing different trainers on a unbalance database.

The second test showed how unbalanced training data produced better models for unbalanced test data, which as mentioned is what IT Relation receives and the LbfgsLogisticRegressionOvA is therefore chosen as the model used in the system. This does however also lead to

²The total amount of correct predictions divided by the total of predictions

³The entire database except the entries used to test

the second hypothesis; training a model on unbalanced data will lead to the model overlooking the smaller categories. To test this the following chapter 10 will take a more in-depth look at the unbalanced `LbfgsLogisticRegressionOvA` model and the balanced `SgdCalibratedOvA` from table 9.9.

10 | Evaluating the Final Model

The project's final model is the unbalanced LbfgsLogisticRegressionOvA (Lbfgs) which has an accuracy of 68.57%. This section will however compare it to the balanced SgdCalibratedOvA (Sgd), which has an accuracy of 55.66% to evaluate whether the different models overlooks smaller categories and discuss why the unbalanced LbfgsLogisticRegressionOvA is the correct to use.

The first step in evaluating the model is to look at the confusion matrix of the models predictions, since plotting the predictions in a confusion matrix gives a better understanding of the models performance. The confusion matrix for the Lbfgs model can be seen in appendix G and the confusion matrix for the Sgd model can be seen in appendix H. From the confusion matrices the three metrics; accuracy, precision and recall can be calculated for each of the models. So far The project has mainly focused on models accuracy since it is the percentage of correct predictions made, but to gain a better understanding of the models their precision and recall should also be evaluated. To calculate precision and recall the following formulas are used:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \quad Recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

Both of these metrics are important to take into consideration when dealing with an unevenly distributed system, since recall tells if smaller categories are being overlooked and precision tells whether or not emails are misclassified[12].

Since this project deals with a multiclass classification task the precision and recall for each class was calculated, see appendix I for the unbalanced Lbfgs model and appendix J for the balanced Sgd model, and used to gather the average. The average precision and recall for the two models is as following:

$$AveragePrecision_{Lbfgs} = 66.67\%$$

$$AveragePrecision_{Sgd} = 50.37\%$$

$$AverageRecall_{Lbfgs} = 62.53\%$$

$$AverageRecall_{Sgd} = 61.77\%$$

By looking at the confusion matrix it is possible to see why the balanced Sgd model performed worse on the unbalanced data than the unbalanced Lbfgs model. The Sgd model has

a bad recall on bigger categories but a good recall on the smaller categories when compared to the Lbfgs model. This results in their average recall being similar but since the accuracy looks at the total amount of correct predictions bigger categories have a higher influence which explains the difference in accuracy.

This is where the second hypothesis mentioned in section 9.4 comes into play, which stated how especially emails belonging to smaller categories would be misclassified. To test this the precision and recall in the unbalanced Lbfgs model of the following categories were looked at; The smallest category, largest category, lowest recall, lowest precision, highest recall, highest precision. The results can be seen below in table 10.1.

Name	Metric	Emails	Precision	Recall
Patch	Highest recall and precision	12,271	94.99 %	93.37%
Desktop Client	Lowest recall and precision	4633	33.33%	4.90%
Sever	Biggest category	70114	83.06 %	87.63%
Browser	Smallest category	3415	54.73%	39.15%

Table 10.1: Precision and Recall of the final model

Overall Judging by the above table and the confusion matrix, there is a clear correlation between the size of a category and the categories recall and precision. This means that to some extent the Lbfgs model is biased towards the categories with more data which supports the second hypothesis. The test however also shows the content of the emails do play a role and it is not just the amount of emails that contributes to the accuracy. This can be seen since the *Patch* category which has the highest precision and recall is also only the 16th largest category, however most of the emails in this category are auto generated and include the words patch or patching¹ which can explain the high precision and recall. The *Desktop client* category which has the lowest recall and precision is the 4th smallest category. The category's emails show that it is a very vague category. The emails are about different problems and the only consistent problems in the category are usually about emails or outlook which have their own categories *Emails* and *Microsoft Office/365* respectively. So if the model detects something about emails it will predict the category to belong to the *Email* category and not *Desktop Client* as seen in the confusion matrix for the Lbfgs model.

The two metrics, precision and recall can further be combined into an average f-score² which

¹Patch and Patching has the same stem.

²See glossary for an explanation of "F-Score".

is a more informative score compared to the accuracy which further allows for either the precision or recall to be valued higher. The f-score for the systems Lbfgs model trained on the unbalanced data is shown below along side the f-score of the Sgd model trained on the balanced data set:

$$F - Score_{Lbfgs} = 2 * \frac{precision * recall}{precision + recall} = 2 * \frac{0.6667 * 0.6253}{0.6667 + 0.6253} = 0.6453$$

$$F - Score_{Sgd} = 2 * \frac{precision * recall}{precision + recall} = 2 * \frac{0.5037 * 0.6177}{0.5037 + 0.6177} = 0.5549$$

By comparing the F-scores, the micro accuracy and macro accuracy it is clear that they support the claim of the unbalanced Lbfgs model being the better model. It does overlook the smaller categories more than the balanced Sdg model but since IT Relation's emails are unbalanced this model overall will lead to less time spent correcting the predictions. The Lbfgs trained on the unbalanced data is therefore the projects final model since it received the highest accuracy of all which is the what IT Relation will be judging the model on. Furthermore it fulfills the projects requirement to create a model with an accuracy of more than 65%.

10.1 | Comparing the System to the Existing System

Below is a complete figure which compares what IT Relations existing system do with the projects system.

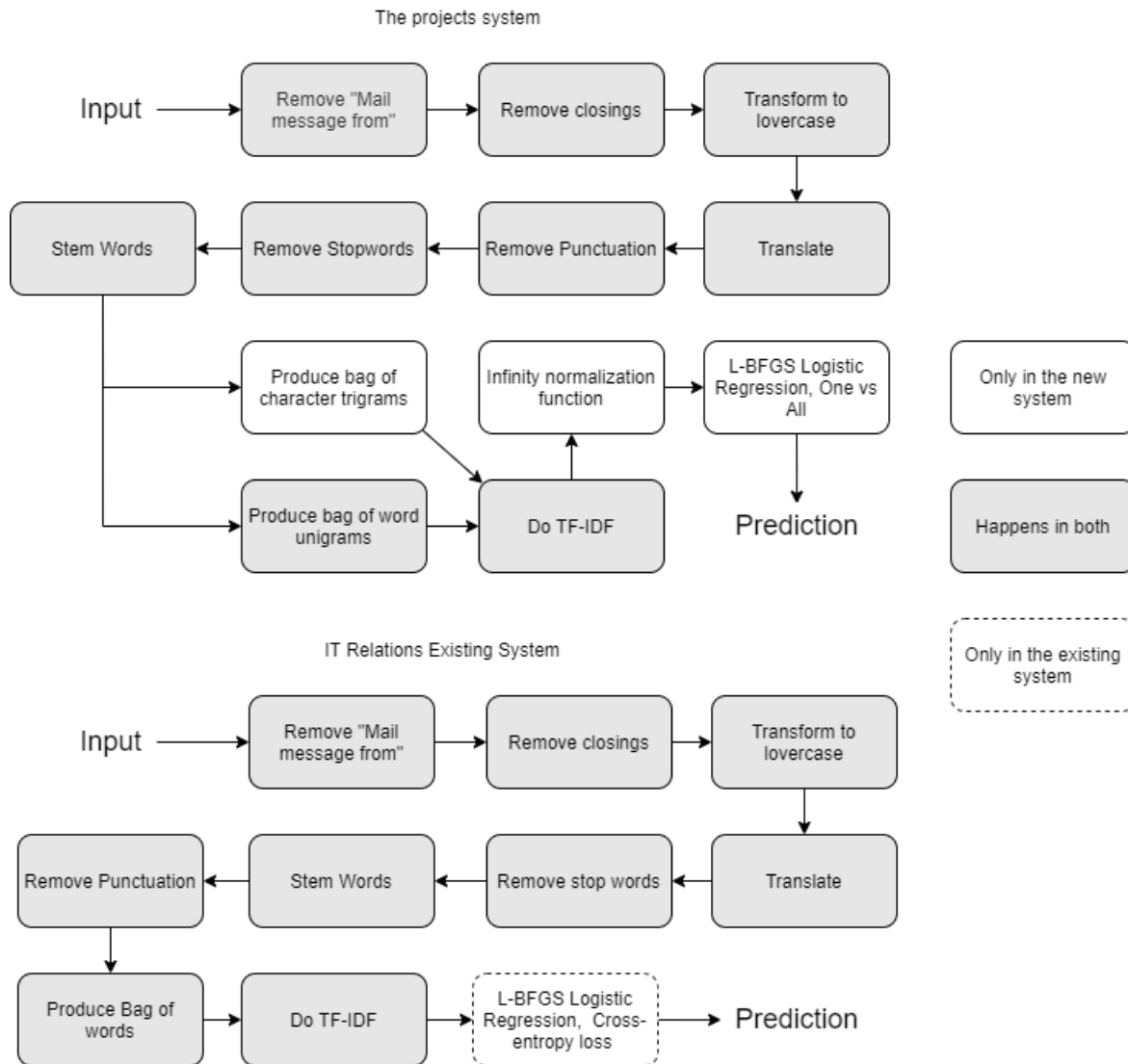


Figure 10.1: Comparing IT Relation's existing system to the projects system

The text pre-processing is similar in both system, which is to be expected since tests showed it to be the best approach. Furthermore a similar stop words list is used in both the projects system and IT Relations system. Both systems also stems words, even though IT Relation does use a different stemmer it is based on the Porter stemmer which the systems Snowball stemmer also is. The biggest difference is near the end. IT Relation split words into unigrams and then use the TF-IDF method to weight the important words more. The projects system also does this but additionally it splits the email up into char trigrams and then use TF-IDF to weight the important char trigrams more. According to the test, shown in subsection 9.2.3, this alone increased the models accuracy by about 2.8%. Another difference is how the projects system normalizes the features with the infinity normalization function, which

IT Relation's system do not. The machine learning algorithms are similar but IT Relation uses a version of the L-BFGS designed to be a multiclass classification algorithm, while this projects use the binary classification version with the One vs All method to solve the multiclass classification problem.

11 | Testing

To develop high-quality software testing is needed in any project to ensure it fulfills its requirements[29, p. 210]. This chapter will explain how the system was tested throughout development, by taking a closer look at the incorporation of a Continuous Integration (CI) workflow as well as unit tests created to ensure main specifications were met.

11.1 | Continuous Integration

This projects system along with any other software system faces the problem of accumulation a technical debt in the form of dependencies, complexity, testing and outside changes which with time can become harder to manage[25]. To prepare against this CI was implemented to automate integration and strengthen the systems quality.

The CI pipeline was created with GitHub actions and ensures all needed dependencies are installed and the code can build without problem every time the main branch is pushed or a pull request is created. This was not only important for the systems development, but also prospectively since the ML models and data are interchangeable. The CI however do not run the systems tests. The CI should ultimately test if the systems tests fails, but since one of the tests access a database they are not incorporated into the CI pipeline since the database should not be accessible for everyone with access to the projects GitHub.

11.2 | Unit Testing

Throughout the project build tests have been the main source of testing. Due to time restrictions, the experimental validation being of a higher priority and taking into consideration that the IT Relations developers seldom interacts with their existing system, it was decided to build test, since it to a certain extent would be acceptable. With the implementations of the user interactions two unit tests were however added to validate requirement F002¹ and F003².

Both unit tests are black box tests, since their objective is to ensure the behavior of the two user interactions *Predict email* and *Pre-process email* function as expected, and both

¹System can classify emails according to their category

²Emails can be pre-processed to remove noise

use the Arrange, Act and Assert pattern³ to prevent test smells.

The first unit test, seen in listing 11, tests if the **ModelFacade** class' *PredictIssue* method returns an actual category. This was chosen since the model and data used can change, but the result should always be a category. The test begins by training a model on a small testing set, since a model is needed to use the *PredictIssue* method. It is not possible to mock a model, since the test depends on an actual prediction from a model. This does make the test prolonged, which is why the TimeoutAttribute from the MSTest Framework is used to ensure it never extends five minutes.

```

1  [TestClass]
2  class MLUnitTest
3  {
4      [TestMethod]
5      [Timeout(300000)] //5min
6      public void TestingCategoryClassification()
7      {
8          // Arrange
9          IDatabase database = new Database();
10         ModelFacade mf = new(database);
11         string email = "Testing email";
12
13         // Act
14         mf.AutoCreateModel("testData.csv", 5);
15         string categoryID = mf.PredictIssue(email, false);
16
17         //Asserts
18         Assert.IsTrue(CategoryList.CategoryValidation(categoryID));
19     }
20 }
```

Listing 11: The *TestingCategoryClassification* unit test in **MLUnitTest**

The second unit test tests the **Preprocessor** class' *Preprocess* method by asserting it produces as expected. The test arranges multiple strings containing different elements the pre-processing handles to then assert if they are pre-processed correctly. This is done since the machine learning model in the system uses a custom pre-processing and therefore subject to change. The second unit test can be seen in appendix K.

³See glossary for an explanation of 'Arrange, Act, Assert'

Considerations were made in terms of validating models accuracy, only allowing to save models with an accuracy fulfilling the reliability requirement. It was however ultimately decided against since it should be IT Relation decision if they want to use models with a lower accuracy or not.

12 | Discussion

This chapter will discuss the projects outcome and the processes of achieving it. It will look at how limited knowledge about ML.NET influenced the project, how IT Relations database can have effected the final accuracy as well as discuss the design elements of the system.

12.1 | Limited Knowledge about ML and ML.NET

At the beginning of the project very little was know about both machine learning and ML.NET. This meant a lot of time not only in the early parts of the project but through out was spent learning about machine learning and experimenting with the ML.NET framework. Because of the evolving knowledge about machine learning and ML.NET the use of AIP became a necessity. AIP's iterative process allowed the project to go back and correct poor implementations as the knowledge about both subjects increased. On the other hand this also means parts of machine learning and ML.NET which could potentially have benefited the project are still unknown. With the projects scope and deadline the outcome is however deemed to be acceptable.

12.2 | Corrupt Data in IT Relations Database

IT Relation's database which was the one used to train the model contained roughly 700.000 emails. These emails are all categorized by service, category and subcategory. Since this is the data our model will use as training data it is assumed to be correct this however might not be the case.

The classification of topics in the database where done by IT Relations existing machine learning model which had an accuracy of 65% which goes against our initial hypothesis of the training data being correct. Furthermore their existing system allows the employees of IT Relation to manually change the service, category or subcategory if they disagree with the models classification before the email enters the database. This results in an uncertainty regarding the classifications shown in the database. In conclusion up to 35% of the emails will have been classified into the wrong topics by the model and there is no way of checking how many and which emails employees manually changed the classification of and if so whether their change was correct. This limit the models ability to improve significantly.

12.3 | Emails can belong to Multiple Categories

Another problem which could have affected the systems accuracy is due to the categories not being clearly separated. Many emails can belong to multiple categories, but the model assume they do not. This means the model's accuracy 68.57% is a very harsh estimate. The model assumes every email which got classified wrongly according to its data is misclassified but in reality that category could also be correct. This is also what caused IT Relation to mention their model is 80% accurate when in reality their model was only 65% accurate. 80% came from them only having an issue with 20% of the emails.

IT Relation agrees and are aware that some emails could belong to a number of categories but only the one chosen by their existing model is shown in the database. Because of this the category the systems model chooses may not be wrong even if it evaluates emails to be incorrectly classified.

12.4 | Pre-processing

ML.NET contains many pre-processing features this project decided not to use since a custom pre-processing was implemented instead. The ML.NET features include, removal of stop words, removal of diacritics, removal of numbers as well as lowercasing. The reason behind using a custom pre-processing was to grant more control over the pre-processing steps and to increase the overall maintainability of the system. By keeping most of the pre-processing to simple methods instead of relaying on a framework it allows for anyone to experiment with changes and not just experienced ML.NET developers which IT Relation's develops are not. However, by not utilising the features that were specifically designed for ML.NET there is a possibility the system will run slower than if the features in ML.NET was implemented instead. However the increase in control and maintainability was deemed more important, since the maintainability of the system was a requirement for the project.

12.5 | The Use of Interfaces in the System

The design decision to use interfaces to access the database was made in order to loosen the systems coupling. The system however does not contain complete separated modules in the form of component-based software and therefore do not utilise the interfaces' full potential. Due to the projects scope it was decided to focus on the machine learning aspect of the project rather than the systems architecture. This is believed to be the correct prioritization for the

project, especially considering the previously mentioned limited knowledge about machine learning and ML.NET.

12.5.1 Danish and English Model

Because only one machine learning model was used it was necessary to translate all emails to the same language. The chosen language was English since it is assumed that translating to English happens often compared to the other languages in the data set. This would mean the machine learning model that does the English translations would be better than for the other languages. However it is still possible that some information gets lost or modified in the translation which could hurt the models accuracy. Therefore it could increase the overall accuracy if a model was created for every language with enough data to train one, which is Danish and English. This method have seen some success before in sentiment analysis where translation caused a 16% drop in accuracy [14]. This project is not a sentiment analysis and the N-Gram test showed the words relation to each other was not important which is the main benefit from not translating. However it is still possible that creating a model for each language could improve the accuracy, if even by a little.

13 | Conclusion

This project resulted in a C# written system using the ML.NET framework. The system has the ability to produce machine learning models one of which uses the L-BFGS Logistic Regression algorithm and the One vs All method to function as a multiclass classifier to accurately predict 68.57% of IT Relations emails.

The projects aim was gathered from meetings with IT Relations where clear definitions of the systems requirements were formed. The most important being the system must be written in C# and able to use multiclass classification to classify IT Relations emails into categories. Furthermore it was gathered how ensuring the systems maintainability would be another focal part for the project, since their existing systems lack there of is the reason for the projects existence. With the requirements in mind the analysis of the system began which resulted in a general overview of the system. This overview will especially be beneficial when the project is to be handed over to IT Relation and integrated with their system.

Additionally to improve the systems maintainability and readability design patterns were used. The Facade pattern was introduced to improve the system by serving as a simple interface to access the complex machine learning part of the system. Furthermore the builder pattern improves the readability as it allows the creation of models to be broken down into a step by step process.

In the implementation phase all the systems functional requirement were realised except requirement F005 *System can classify emails according to their service* since emails are classified by their category, which then in tern automatically produces the emails service. Furthermore the four guidelines from better code hub helped minimize code smells which added to the codes maintainability. The system is able to pre-process text by removing noise in the form of auto generated text, stop words as well as stemming words with the use of the Snowball stemmer. The system is furthermore capable of training and testing a model based data from a .csv database in about 55 minutes, which is much less than the desired time of 4 hours as stated in the performance requirement. The model also has an accuracy of 68.57% which is 3.57% higher than the model IT Relation currently uses at 65% which fulfills the reliability requirement. At the moment the model only predicts on the biggest 35 categories, but this can be changed by training a new model on a data set with more categories. It can therefore be concluded the project fulfills its aim since the projects system is a more maintainable and accurate solution for IT Relation to classify their emails than their existing system. The system could potentially be further developed which will be mentioned in the

following section 13.1, but can at the moment be integrated with IT Relation's system and after implementing their database the system will function as required.

13.1 | Future Work

13.1.1 Retrain

It was discussed adding functionality that allowed models to be retrained. This would mean taking a model and use that as base to continue training. However ML.NET have only implemented this feature on one multiclass classification trainer, the LbfgsMaximumEntropy. Therefore it was not seen as something worth spending time on.

13.1.2 Multi-label Classification

Multi-label classification is similar to multiclass classification, the main difference being how in multi-label classification multiple labels are assign to an email rather than just predicting one category. Because some of IT Relation's emails can fit into multiple categories multi-label classification could produce a more accurate representation of the systems accuracy. In the project the systems model may predict wrongly and count the email as misclassified, but if a human received the email they may not believe the model misclassified it. Because of this multi-label classification would be a good solution to enhance the systems accuracy.

13.1.3 Subcategory

For this project it was decided to classify based on the email category, since it would automatically also classify the emails service. IT Relations system does however, as mentioned in section 5.1, also contain sub categories which has been neglected in this project. The system could therefore be extended to also look at subcategories. This could be done by either finding the category first and then predict between the categories sub categories or the model could try and predict the subcategory which would then also grant the category.

Bibliography

- [1] Ahmed, Z. *et al.*, “Machine learning at microsoft with ML .net,” *CoRR*, vol. abs/1905.05715, 2019. [Online]. Available: <http://arxiv.org/abs/1905.05715>.
- [2] Arlow, J. and Neustadt, I., *UML 2 and the unified process*, 2nd. Pearson Education, 2005, ISBN: 9780132702638.
- [3] Berry, L., “Customer support services’ next horizon: A commentary,” *European journal of marketing*, vol. 54, no. 7, pp. 1805–1806, 2020. [Online]. Available: <https://www.emerald.com/insight/content/doi/10.1108/EJM-07-2020-971/full/html>.
- [4] Cadariu, M., “Higher grades for better code: A proposed add-on for software engineering courses,” *Medium*, 2016. [Online]. Available: <https://medium.com/bettercode/higher-grades-for-better-code-23183648f793>.
- [5] Chazdon, R. *et al.*, “A novel statistical method for classifying habitat generalists and specialists,” *Ecology*, vol. 92, no. 6, pp. 1332–1343, 2011. [Online]. Available: <https://www.jstor.org/stable/23035004>.
- [6] Donges, N., “Gradient descent: An introduction to 1 of machine learning’s most popular algorithms,” [Online]. Available: <https://builtin.com/data-science/gradient-descent>.
- [7] Fowler, M., *Refactoring: Improving the Design of Existing Code*, 2nd. Pearson Education, 2019, ISBN: 9780134757599.
- [8] GeeksForGeeks, “Introduction to stemming,” 2020. [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-stemming/>.
- [9] Gershon, A. *et al.*, “Newton raphson method,” [Online]. Available: <https://brilliant.org/wiki/newton-raphson-method/>.
- [10] Ignatow, G. and Mihalcea, R., “An introduction to text mining: Research design, data collection, and analysis,” in. SAGE Publications, Inc, 2018, ch. 13: Text Classification, pp. 171–186. [Online]. Available: <http://www.doi.org.proxy1-bib.sdu.dk:2048/10.4135/9781506336985>.
- [11] Jing, H., “Why linear regression is not suitable for classification,” [Online]. Available: <https://jinglescode.github.io/2019/05/07/why-linear-regression-is-not-suitable-for-classification>.
- [12] Jordan, J., “Evaluating a machine learning model,” 2017. [Online]. Available: <https://www.jeremyjordan.me/evaluating-a-machine-learning-model/>.

- [13] Kerievsky, J., *Refactoring to patterns*, 1st. Addison-Wesley Professional, 2004, ISBN: 9780321213358.
- [14] Koning, M. de, “To translate or not to translate, best practices in non-english sentiment analysis,” [Online]. Available: <https://towardsdatascience.com/to-translate-or-not-to-translate-best-practices-in-non-english-sentiment-analysis-144a53613913>.
- [15] Majumder, P. *et al.*, “Statistical vs. rule-based stemming for monolingual french retrieval,” *Springer, Berlin, Heidelberg*, vol. 4730, pp. 107–110, 2007. [Online]. Available: https://doi-org.proxy1-bib.sdu.dk/10.1007/978-3-540-74999-8_14.
- [16] Martin, R., *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st. Prentice Hall, 2007, ISBN: 9780132350884.
- [17] Microsoft, “What is ml.net?,” [Online]. Available: <https://dotnet.microsoft.com/learn/ml-dotnet/what-is-mldotnet>.
- [18] Najafabadi, M. M. *et al.*, “Large-scale distributed l-bfgs,” *J Big Data*, vol. 4, no. 22, 2017. [Online]. Available: <https://doi.org/10.1186/s40537-017-0084-5>.
- [19] Nanda, J., “Scene recognition with bag of words,” [Online]. Available: https://www.cc.gatech.edu/classes/AY2016/cs4476_fall/results/proj4/html/jnanda3/index.html/.
- [20] Nikam, S., “A comparative study of classification techniques in data mining algorithms,” *Orient. J. Comp. Sci. and Technol*, vol. 8, 2015. [Online]. Available: <http://www.computerscijournal.org/?p=1592>.
- [21] Pöyhönen, S. *et al.*, “Coupling pairwise support vector machines for fault classification,” *Control Engineering Practice*, vol. 13, no. 6, pp. 759–769, 2005. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S09670666104001789>.
- [22] Rajlich, V., *Software Engineering: The Current Practice*, 1st. Chapman and Hall/CRC, 2011, ISBN: 9781439841228.
- [23] refactoringguru, “Builder,” [Online]. Available: <https://refactoring.guru/design-patterns/builder>.
- [24] refactoringguru, “Facade,” [Online]. Available: <https://refactoring.guru/design-patterns/facade>.
- [25] Sato, D. *et al.*, “Continuous delivery for machine learning,” 2019. [Online]. Available: <https://martinfowler.com/articles/cd4ml.html#TechnicalComponentsOfCd4ml>.
- [26] Swaminathan, S., “Logistic regression — detailed overview,” [Online]. Available: <https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc/>.
- [27] Taipale, K., “Optimization and newton’s method,” [Online]. Available: https://www.softcover.io/read/bf34ea25/math_for_finance/multivariable_methods.

- [28] Tokuç, A., “Gradient descent equation in logistic regression,” [Online]. Available: <https://www.baeldung.com/cs/gradient-descent-logistic-regression>.
- [29] Tsui, F. and Karam, O., *Essentials of Software Engineering*, 4th. Jones & Bartlett Learning, 2016, ISBN: 9781284106077.
- [30] Yawen, W. *et al.*, “Research of email classification based on deep neural network,” *International journal of advanced network, monitoring, and controls*, vol. 3, 2. [Online]. Available: <https://doi.org/10.21307/ijanmc-2018-038>.

Appendices

A | BetterCodeHub Guidelines

1) Write Short Units of Code This guideline states that no code unit should extend the length of 15 lines of code. This can be done by either writing short units of code to begin with or dividing longer units into multiple smaller units. This is important since short units are easier to test, analyse and reuse.

2) Write Simple Units of Code this guideline means that no unit should have more than 4 branch points per unit. Branch points are counted for every if, switch case, for loop, && or || in the unit. This improves maintainability since less branch point will in most cases make a unit easier to understand and also to modify or test.

3) Write Code Once The third guideline deals with duplicated code, saying that no duplicated code should be present since it is error-prone because it is more difficult to maintain. One way to avoid this is by extracting methods.

10) Write Clean Code This guideline builds on the idea that you should always leave code cleaner than you found it and stems from the goal of not leaving code smells behind to begin with. This guideline improves maintainability since clean code is always more maintainable[13, p. 10]. To further detail this guideline the following 7 rules were used during this project.

1. Leave no unit-level code smells behind
2. Leave no bad comments behind
3. Leave no code in comments behind
4. Leave no dead code behind
5. Leave no long identifier names behind
6. Leave no magic constants behind
7. Leave no badly handled exceptions behind

The rest of the guidelines was not included in this project and will therefore not be described.

B | Use Case Descriptions

Use case	Description
Balance database	Create a new database which only contains a given amount of entries per category
Clean database	Clean a database, by removing incorrect columns
Clear	Clears the command prompt
Evaluate model based on a database	Evaluate the accuracy of a model with a given database
Train model	Train a model based on a given database
Exit	Stop the system
Help	All possible interactions are printed as a guide
Predict email	Predict a category of a given email
Pre-process email	Pre-process a given text
Remove small categories from database	Remove all categories with less than a chosen amount of entries from a database
Save/Load model	Save or a trained model
Split database	Splitting a given database into 80% training data and 20% testing data.
Update stop words list	Switch out the stop words list with a given .txt file

C | Detailed Use Case: Train Model

Detailed Use Case: Train Model
ID: 01
Primary actor: User
Secondary actor: None
Short description: User trains a model based on a given database
Preconditions: System is running and a database containing emails and their categories is available
Main flow: <ol style="list-style-type: none"> 1. User types in the 'train' command in the command line. 2. If command is used by typing 'train' followed by a database path. A model is created by: <ol style="list-style-type: none"> 1) Getting training data from the given database 2) Transforming the training data by pre processing 3) Training the model 4) Evaluating the model 3. Else see alternative flow
Post conditions: A machine learning model is build
Alternative flow: Starts after 2. in main flow <ol style="list-style-type: none"> 1. If command is used by typing 'train' followed by a database path and an integer illustrating the time to train A model is created by: <ol style="list-style-type: none"> 1) Getting training data from the given database 2) Transforming the training data by pre processing 3) Training different models in the given time interval 4) Evaluating the models and choosing the best to keep 2. Else command is not recognized <ol style="list-style-type: none"> (a) An error message is displayed to the user
Notes: The program keeps looking for a command until: <ol style="list-style-type: none"> 1. The 'exit' command is used to close the system 2. The system is manually closed

D | SGD and L-BFGS

Stochastic Gradient Decent

Gradient descent is a simple machine learning algorithm but also one of the most popular[6]. In gradient descent the goal is to find values to the weights w that minimise the square difference between the functions predictions and the actual value for every data in the data set m . The notation $f_w(x)$ means that the function $f(x)$ depend on the values of the weights w .

$$\text{Minimize}_w \rightarrow g(w) = \sum_{i=1}^m (f_w(x^i) - y^i)^2$$

The above function is called the cost function $g(w)$ and the gradient decent algorithm will try to find the values for w that will minimize the function. This works in linear regression because as it turns out there are no local minimum only a global minimum to this function. However when using logistic regression local minimum do appear so if gradient decent is used it might get 'stuck' in a local minimum it wants to improve. Instead when using logistic regression the cost function below is used[28].

$$\text{Minimize}_w \rightarrow g(w) = -\frac{1}{m} \sum_{i=1}^m y^i * \log(f_w(x^i)) + (1 - y^i) * \log(1 - f_w(x^i))$$

This removes any local minimum which allows gradient decent to find the global minimum. Gradient decent starts at a random w values and then each iteration looks at values for w 's that are nearby and if a value decreases the cost function $g(w)$ it is used instead and a new iteration begins. Because it only looks at nearby values the change in w will be small, however you can change the size of the steps, which is referred to as the learning rate. The gradient decent continues until it reaches a point where any change to w will increase the function $g(w)$, meaning it is at a global minimum. The w values can then be saved and used in the original prediction function $f(x)$.

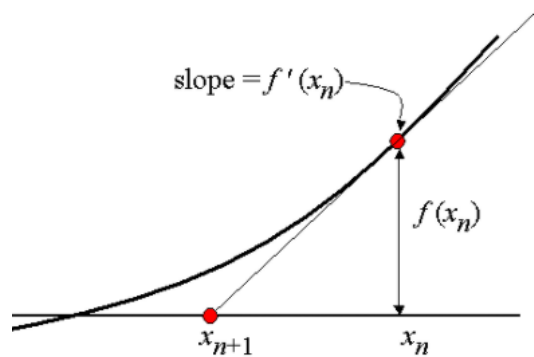
To calculate the above cost function the algorithm have to look at every data in the data set, this is referred to as Batch Gradient decent. For large data sets this simply takes to long so a different version called SGD have been created. This method does not use every data in the data set to calculate the cost function $g(w)$ but picks 1 random data each iteration. Because of this each step the algorithm takes are not guaranteed to be the best but over time it will gradually move the right way. It is also not guaranteed to find the global minimum but it will usually get close enough.

Limited-memory Broyden–Fletcher–Goldfarb–Shanno

L-BFGS is based on Newtons method which is an algorithm used to approximate roots of a

function $f(x)$. It does this by starting with a guess x_0 and by creating a tangent using the derivative to the function at the guess. Where that tangent intersects the $y = 0$ line becomes the new guess x_1 . This continues until the difference between x_t and x_{t+1} becomes small. Formally written as [9]:

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$



Source: [9]

Figure 1: Newtons Method

Finding a minimum to a function does not require finding an x value where $f(x) = 0$. However it is possible to take the derivative of the function and use that to find the minimum, since $f'(x) = 0$ the x value where no y change is happening in the original $f(x)$. If no y change is happening it is either a global minimum or a global maximum function. This means we can use the cost functions and estimate the minimum using the method [27]:

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}$$

If x is in 1 dimension it is possible to use the first and second order derivative as above. But when x is in higher dimensions the multivariable equivalent of the first and second order derivative is used instead called the gradient and the hessian respectively. The inverse hessian is used but the idea is the same. [27]:

$$x_{t+1} = x_t - (f''(x_t))^{-1} * f'(x_t)$$

The above method of finding the minimum of $f(x)$ is how the BFGS algorithm works. The algorithm is however a part of the Quasi-Newton family which means it uses an estimation instead of calculating the values of the inverse hessian $(f''(x_t))^{-1}$. The L-BFGS uses the same method as BFGS but while BFGS stores a $n \times n$ approximation of the inverse hessian the L-BFGS only stores few values using the history of past x values. This makes it well suited for problems with a high number of variables n [18].

E | General Pre-processing: Explanations

Lowercase

To ensure uppercase versions of words are not treated differently than a lowercase version everything is transformed to lowercase. This is done automatically in ML.NET, but to make the other pre-processing steps easier it was decided to also do it in the custom pre-processing before anything else.

Remove closings

IT Relations ticketing system does not always separate the email from its correspondence, meaning the email will also contain the entire previous communication. To remove this as well as any signature that might be used the system check for both Danish and English closings and removes everything written from that point.

Remove "Mail message from"

The ticketing system also add "Mail Message From:" at some emails as well as the day and time of the email being received. This does not add useful information as is therefore removed.

Remove punctuation

This removes commas and periods which helps the model to treat words at the end of sentences the same as other words.

F | Four Other General Pre-processing Methods

The following are descriptions of each of the four extra general pre-processing methods which were tested but not used due to the test results showing they harmed the models accuracy.

Remove numbers

Remove all numbers from a email.

Remove diacritics

Since words with diacritics would be seen as different words removing the diacritics would solve that problem.

Remove symbols

Remove all symbols from a email.

Remove Apostrophe

Removes all apostrophes from a email

The results from adding each of these steps individually to the general pre-processing can be seen below.

Option	Accuracy	Training time
Without the four	63.97 %	142,644 ms
with remove numbers	62.45 %	194,403 ms
with remove diacritics	63.53 %	181,461 ms
with remove symbols	63.26 %	144,566 ms
with remove apostrophe	63.57 %	172.381 ms

Note: Training time is both pre-processing time and time to train the model.

G | Confusion Matrix: LbfgsLogisticRegressionOvA

This is the Confusion matrix for the LbfgsLogisticRegressionOvA model trained on the unbalanced data set

True ↓ / Predicted →	0	1	2	3	4	5	6	7	8	9	10	11
0: Microsoft Office/365	2634	41	281	581	15	3	12	26	20	53	14	182
1: ERP	15	3109	179	24	2	44	2	3	12	32	12	113
2: Account/Right	182	237	7270	309	17	43	107	29	32	133	48	269
3: E-Mail	418	31	281	2213	20	0	12	109	9	17	24	65
4: Consultancy	51	5	39	43	423	2	15	5	10	3	2	56
5: POS Computer	2	36	32	2	0	1079	2	0	2	7	3	29
6: VPN	5	12	140	2	3	0	786	0	9	15	7	34
7: Spamfilter	12	9	27	153	0	0	0	1065	9	0	12	7
8: Server *	7	22	29	14	10	12	0	12	8900	20	56	41
9: Terminalserver	77	27	153	12	5	10	26	2	49	1374	2	123
10: Double Incident	20	60	84	43	9	7	9	14	121	10	915	53
11: Other	286	213	360	106	32	41	55	12	61	128	53	1931
12: Telephony	48	7	17	7	2	2	2	7	2	2	0	44
13: General	155	136	373	152	65	27	26	145	87	53	218	278
14: Citrix	66	43	133	20	5	2	9	2	58	63	5	130
15: User Administration	90	34	457	244	14	0	9	0	3	15	20	53
16: Cashpoint	2	3	15	0	0	12	0	0	2	2	0	9
17: Server	26	46	85	29	27	7	15	7	433	87	34	107
18: Website	14	34	82	15	15	9	5	5	56	3	22	131
19: Peripheral Equipment	20	2	29	10	0	10	2	0	12	5	9	51
20: Creditcard	0	3	5	0	0	113	0	0	2	0	0	7
21: Desktop Client	247	5	48	211	0	0	0	3	0	9	2	20
22: Computer	131	17	152	38	19	31	32	3	15	85	7	194
23: Fileshare	17	7	218	10	2	3	19	3	10	9	7	38
24: Browser	5	2	31	7	0	3	0	2	0	3	5	116
25: Backup/restore	17	3	17	14	19	2	2	0	46	3	5	22
26: Network Printer	10	19	29	5	2	15	0	0	9	14	7	41
27: Conventus	7	2	26	10	0	2	0	0	0	0	0	10
28: Network	14	9	38	12	26	2	19	3	17	17	7	27
29: Patch	2	0	2	0	2	0	0	0	31	0	3	32
30: SMS Passcode	3	2	107	0	0	0	9	0	2	22	2	3
31: Sikker Mail	3	2	10	34	2	0	2	10	12	0	5	7
32: Internal Sales	17	12	22	7	24	2	0	2	2	0	5	19
33: Network *	3	2	3	3	5	2	7	2	215	7	12	7
34: Storage	0	0	5	0	0	0	2	2	469	0	9	2

* (Monitoring)

Continuation of previous table

True ↓ / Predicted →	12	13	14	15	16	17	18	19	20	21	22
0: Microsoft Office/365	14	155	39	106	0	15	5	2	0	31	72
1: ERP	2	75	29	26	0	15	5	2	0	2	7
2: Account/Right	17	314	82	360	3	17	26	14	10	0	131
3: E-Mail	2	176	17	218	0	7	7	3	0	12	26
4: Consultancy	3	106	9	32	0	55	7	2	2	0	29
5: POS Computer	0	39	5	2	12	3	3	5	99	0	15
6: VPN	0	49	3	15	2	10	3	0	0	0	27
7: Spamfilter	2	227	2	7	0	0	5	2	0	2	9
8: Server *	0	172	19	12	2	220	12	10	0	0	17
9: Terminalserver	0	38	70	17	0	46	3	0	0	2	43
10: Double Incident	5	327	15	48	2	12	9	3	0	0	10
11: Other	49	363	97	66	14	82	55	19	5	2	218
12: Telephony	278	53	0	51	3	2	0	19	2	2	5
13: General	29	5797	73	215	106	66	53	22	3	0	107
14: Citrix	0	53	1558	14	0	29	5	3	0	2	27
15: User Administration	32	131	15	4543	0	3	0	5	0	3	44
16: Cashpoint	2	43	0	0	1234	2	0	3	2	0	7
17: Server	0	135	43	24	0	992	29	2	0	0	19
18: Website	2	118	7	24	2	12	331	0	2	0	5
19: Peripheral Equipment	51	34	9	20	3	3	0	332	2	0	148
20: Creditcard	0	3	0	0	3	0	0	3	326	0	5
21: Desktop Client	0	12	10	5	0	0	0	0	0	31	9
22: Computer	5	90	38	85	2	14	2	94	0	5	1294
23: Fileshare	2	36	12	32	0	5	5	2	0	0	36
24: Browser	0	36	10	2	0	3	43	0	0	0	14
25: Backup/restore	0	32	3	14	0	24	0	0	0	0	2
26: Network Printer	0	34	10	14	0	5	2	5	0	0	17
27: Conventus	0	38	0	2	0	0	3	0	2	0	0
28: Network	3	107	17	15	5	22	3	7	0	0	27
29: Patch	0	12	2	3	0	20	0	0	0	0	2
30: SMS Passcode	2	22	19	29	0	0	0	0	0	0	3
31: Sikker Mail	0	65	0	2	0	7	2	0	0	0	0
32: Internal Sales	7	51	0	14	0	2	3	31	0	0	24
33: Network *	0	58	3	2	0	14	3	0	0	0	3
34: Storage	0	14	0	0	0	31	0	2	0	0	2

* (Monitoring)

Continuation of previous table

True ↓ / Predicted →	23	24	25	26	27	28	29	30	31	32	33	34
0: Microsoft Office/365	24	2	10	15	2	2	2	3	3	3	0	5
1: ERP	0	0	9	20	0	2	0	0	0	2	0	0
2: Account/Right	176	2	10	24	0	17	0	87	5	2	2	0
3: E-Mail	2	0	9	9	2	2	0	2	34	0	0	0
4: Consultancy	7	0	19	26	0	29	0	0	7	17	0	0
5: POS Computer	2	0	0	31	2	10	0	0	0	0	0	0
6: VPN	14	0	0	5	0	14	0	12	0	0	3	0
7: Spamfilter	0	0	0	0	0	0	0	0	14	3	0	2
8: Server *	10	2	49	3	0	3	44	5	2	3	131	315
9: Terminalserver	22	3	3	29	0	15	2	9	3	0	2	0
10: Double Incident	3	0	17	14	0	12	2	3	0	2	19	2
11: Other	51	56	17	53	10	15	7	22	7	5	0	0
12: Telephony	3	0	0	2	0	14	0	2	0	2	0	0
13: General	27	24	20	43	38	58	3	9	27	22	20	3
14: Citrix	9	2	2	15	0	2	2	15	0	0	2	2
15: User Administration	26	0	2	5	0	0	0	15	3	2	2	2
16: Cashpoint	0	0	0	2	2	3	0	0	0	0	0	0
17: Server	31	0	60	17	0	12	17	3	7	0	12	61
18: Website	2	53	3	3	2	0	0	0	2	0	2	0
19: Peripheral Equipment	0	2	0	17	0	7	0	0	0	5	0	0
20: Creditcard	0	2	0	0	0	0	0	0	0	0	0	0
21: Desktop Client	5	0	0	5	0	3	0	0	0	0	0	0
22: Computer	49	9	7	48	0	24	3	0	2	5	2	3
23: Fileshare	574	0	7	2	0	5	0	0	0	0	0	2
24: Browser	2	188	0	5	0	3	0	0	0	0	0	0
25: Backup/restore	19	0	1561	3	0	5	0	0	2	2	0	0
26: Network Printer	2	0	0	1435	0	10	0	0	0	2	0	0
27: Conventus	0	0	0	0	847	0	0	0	0	0	0	0
28: Network	5	0	2	10	2	465	0	0	0	0	48	0
29: Patch	0	0	2	0	0	0	1584	0	0	0	0	0
30: SMS Passcode	0	0	0	0	0	0	0	263	0	0	0	0
31: Sikker Mail	0	0	3	0	0	2	0	0	1846	0	0	0
32: Internal Sales	0	0	2	0	0	2	0	0	3	409	2	0
33: Network *	0	0	3	3	0	36	0	2	0	2	581	9
34: Storage	2	0	7	0	0	0	2	0	0	0	2	610

* (Monitoring)

H | Confusion Matrix: SgdCalibratedOvA

This is the Confusion matrix for the SgdCalibratedOvA model trained on the balanced data set.

True ↓ / Predicted →	0	1	2	3	4	5	6	7	8	9	10	11
0: Microsoft Office/365	1.757	27	108	337	114	8	24	72	9	96	97	148
1: ERP	25	2.819	75	19	18	103	17	21	1	67	186	96
2: Account/Right	222	237	5.144	412	141	84	389	83	4	274	268	250
3: E-mail	215	22	107	1.725	75	5	8	333	2	22	135	62
4: Cosultancy	21	8	9	39	510	1	29	19	3	17	31	33
5: POS Computer	5	15	9	0	2	978	1	1	1	6	11	14
6: VPN	8	14	22	1	12	4	931	1	2	22	12	15
7: Spamfilter	9	5	9	111	17	0	0	1.072	3	1	53	17
8: Server*	2	19	6	6	20	4	4	3	7.381	90	170	45
9 Terminalserver	45	28	65	18	29	15	51	3	4	1.364	24	59
10: Double Incident	33	43	29	27	45	9	23	64	64	11	634	26
11: Other	162	156	125	96	133	78	112	92	19	165	222	1.086
12: Telephony	10	2	3	4	3	3	1	2	2	4	10	9
13: General	346	64	135	129	186	46	27	758	65	32	1.076	152
14: Citrix	23	62	63	15	28	8	18	4	82	71	54	61
15 User Administration	166	52	264	291	167	6	45	65	4	39	204	51
16: Cashpoint	0	2	2	1	1	19	1	0	0	8	9	5
17: Server	12	15	9	15	58	9	7	10	403	61	77	64
18: Website	3	9	20	6	22	24	6	9	23	13	25	37
19: Peripheral Equipment	5	3	7	4	5	10	2	3	0	10	13	9
20: Creditcard	0	0	0	0	0	45	0	0	0	0	0	3
21: Desktop Client	247	1	9	98	18	1	5	12	1	21	8	11
22: Computer	69	11	51	12	62	15	50	14	2	99	42	92
23: Fileshare	18	4	17	9	7	0	33	2	2	16	14	13
24: Browser	8	0	2	1	1	3	0	0	1	6	0	13
25: Backup/restore	8	1	13	7	10	1	1	2	75	3	14	7
26: Network Printer	5	10	11	8	9	28	8	8	3	28	22	21
27: Conventus	10	8	28	12	10	9	0	1	0	5	1	12
28: Network	7	3	6	4	42	16	31	4	6	20	21	10
29: Patch	1	0	22	0	3	0	18	0	0	1	1	4
30: SMS Password	1	1	0	0	7	0	0	3	4	1	2	8
31: Sikker Mail	7	0	5	53	13	1	0	26	4	1	31	21
32: Internal Sales	18	8	2	6	100	7	3	6	6	2	15	16
33: Network*	0	0	0	17	7	0	4	0	127	2	27	4
34: Storage	0	0	1	1	0	0	1	1	266	0	1	3

* (Monitoring)

Topic Classification

Continuation of previous table

True ↓ / Predicted →	12	13	14	15	16	17	18	19	20	21	22
0: Microsoft Office/365	99	85	81	67	8	25	35	81	2	736	109
1: ERP	36	10	38	32	4	37	73	59	5	12	39
2: Account/Right	119	215	176	475	15	69	176	44	6	138	218
3: E-Mail	50	68	18	159	8	15	44	19	1	543	39
4: Consultancy	11	13	9	28	1	23	29	13	1	15	37
5: POS Computer	4	5	2	1	16	3	41	26	313	2	15
6: VPN	4	4	9	9	0	7	10	2	2	3	24
7: Spamfilter	16	78	1	8	3	2	17	4	0	27	0
8: Server*	3	91	78	10	2	326	114	0	0	3	10
9: Terminalserver	10	20	71	22	6	72	22	28	4	27	73
10: Double Incident	34	204	13	55	10	88	226	15	3	11	16
11: Other	159	102	140	74	15	118	223	148	21	100	295
12: Telephony	460	16	3	9	4	1	3	52	0	4	2
13: General	251	2.795	39	171	154	88	237	57	15	74	103
14: Citrix	7	7	1.516	16	6	32	32	8	1	45	20
15: User Administration	268	193	24	3.524	5	18	45	57	1	36	80
16: Cashpoint	7	9	0	1	1.318	1	4	3	16	1	1
17: Server	2	27	26	10	3	462	90	7	0	11	11
18: Website	10	13	7	4	11	10	400	3	2	1	15
19: Peripheral Equipment	75	9	6	3	3	7	2	478	4	2	65
20: Creditcard	4	1	0	0	4	1	0	1	484	0	4
21: Desktop Client	6	2	15	7	0	4	2	8	0	331	24
22: Computer	27	37	31	47	13	10	13	315	4	31	1.253
23: Fileshare	2	2	8	8	1	10	9	5	1	3	27
24: Browser	3	1	11	0	2	0	34	0	0	2	9
25: Backup/restore	1	2	0	87	0	25	9	2	0	5	5
26: Network Printer	7	4	9	4	2	2	3	16	3	2	14
27: Conventus	10	7	4	12	19	2	24	1	6	6	4
28: Network	19	14	5	6	4	11	6	13	6	7	25
29: Patch	6	0	8	1	1	0	1	0	0	1	1
30: SMS Passcode	0	2	5	1	0	17	5	1	0	0	3
31: Sikker Mail	7	26	1	1	4	15	11	0	0	14	5
32: Internal Sales	23	27	4	4	4	6	21	59	3	7	43
33: Network*	2	9	2	2	0	8	5	1	0	1	4
34: Storage	0	1	2	0	1	19	2	0	0	0	3

* (Monitoring)

Topic Classification

Continuation of previous table

True ↓ / Predicted →	23	24	25	26	27	28	29	30	31	32	33	34
0: Microsoft Office/365	99	49	16	28	21	20	31	2	10	19	3	13
1: ERP	27	11	5	42	7	18	11	1	2	10	3	1
2: Account/Right	578	111	12	30	21	46	472	5	21	11	10	5
3: E-Mail	30	22	18	26	10	13	10	1	107	8	2	1
4: Consultancy	11	4	17	8	2	52	6	8	8	47	4	4
5: POS Computer	1	2	0	30	0	9	1	0	0	2	2	0
6: VPN	41	4	2	4	1	25	38	1	0	1	4	0
7: Spamfilter	4	5	10	2	0	3	3	0	33	5	6	7
8: Server*	7	2	76	17	0	24	6	52	1	10	656	1.280
9: Terminalserver	34	27	4	35	2	30	43	2	1	0	5	4
10: Double Incident	24	2	35	15	1	13	6	1	45	15	21	15
11: Other	127	334	19	82	14	44	50	31	15	29	14	9
12: Telephony	0	2	3	0	0	6	5	0	0	2	1	0
13: General	465	84	41	21	68	98	22	24	475	138	335	21
14: Citrix	9	41	8	25	2	15	47	5	1	1	12	39
15: User Administration	109	16	9	14	12	15	88	1	15	71	4	1
16: Cashpoint	0	2	1	4	3	6	0	0	0	0	0	0
17: Server	36	11	42	11	0	42	5	392	6	3	206	315
18: Website	5	243	4	6	7	7	7	0	4	7	5	4
19: Peripheral Equipment	4	3	1	15	1	10	3	1	0	9	2	0
20: Creditcard	0	0	0	3	0	1	0	0	0	0	0	0
21: Desktop Client	1	3	0	2	1	2	0	0	3	0	0	0
22: Computer	91	46	11	31	4	61	11	6	4	46	3	5
23: Fileshare	842	1	22	3	1	6	2	0	0	0	1	5
24: Browser	1	404	1	5	1	2	0	0	1	1	0	0
25: Backup/restore	21	1	1.561	0	0	5	0	1	0	0	12	15
26: Network Printer	6	5	0	1.463	0	19	2	0	3	4	8	3
27: Conventus	4	27	1	2	751	3	4	0	3	0	1	0
28: Network	8	7	1	12	0	576	2	1	3	4	30	0
29: Patch	1	4	0	1	0	0	447	0	0	0	0	0
30: SMS Passcode	0	1	3	0	0	0	0	1.767	0	0	3	6
31: Sikker Mail	2	3	6	1	2	5	4	5	1.810	26	2	5
32: Internal Sales	2	4	2	4	2	18	2	1	7	223	2	0
33: Network*	0	2	2	1	0	53	0	0	76	0	553	113
34: Storage	5	0	1	0	0	1	0	0	0	0	98	820

* (Monitoring)

I | Precision and Recall: LbfgsLogisticRegressionOvA

This is the precision and recall for each category in the LbfgsLogisticRegressionOvA model trained on the balanced data set

Category	Recall	Precision
Microsoft Office/365	0,6023	0,5716
ERP 2	0,8310	0,7421
Account/Rights	0,7289	0,6744
E-Mail	0,5943	0,5112
Consultancy	0,4218	0,5536
POS Computer	0,7590	0,7276
VPN	0,6710	0,6652
Spamfilter	0,6801	0,7234
Server (Monitoring)	0,8763	0,8306
Terminalserver	0,6341	0,6267
Double Incident	0,4954	0,5980
Other	0,4300	0,4543
Telephony	0,4837	0,5488
General	0,6835	0,6432
Citrix	0,6846	0,7031
User Administration	0,7868	0,7550
Cashpoint	0,9188	0,8862
Server (Servers, Storage, Data)	0,4193	0,5706
Website	0,3446	0,5301
Peripheral Equipment	0,4239	0,5620
Creditcard terminal	0,6895	0,7180
Desktop Client	0,0490	0,3333
Computer	0,5167	0,5383
Fileshare	0,5409	0,5392
Browser	0,3915	0,5473
Backup/restore	0,8593	0,8561
Network Printer	0,8514	0,7782
Conventus	0,8939	0,9360
Network (Communication)	0,5009	0,6026
Patch	0,9337	0,9499
SMS Passcode	0,5385	0,5811
Sikker Mail	0,9170	0,9385
Internal Sales	0,6202	0,8392
Monitoring	0,5889	0,7016
Storage	0,5272	0,6007
Average	0,6253	0,6667

J | Precision and Recall: SgdCalibratedOvA

This is the precision and recall for each category in the SgdCalibratedOvA model trained on the balanced data set

Category	Recall	Precision
Microsoft Office/365	0,3961	0,5292
ERP 2	0,7173	0,7725
Account/Rights	0,4908	0,8065
E-Mail	0,4397	0,4951
Consultancy	0,4762	0,2720
POS Computer	0,6443	0,6351
VPN	0,7514	0,5032
Spamfilter	0,7002	0,3979
Server (Monitoring)	0,7017	0,8614
Terminalserver	0,6070	0,5291
Double Incident	0,3380	0,1806
Other	0,2356	0,4384
Telephony	0,7348	0,2641
General	0,3179	0,6819
Citrix	0,6359	0,6418
User Administration	0,5913	0,7254
Cashpoint	0,9249	0,8002
Server (Servers, Storage, Data)	0,1880	0,3012
Website	0,4115	0,2033
Peripheral Equipment	0,6176	0,3132
Creditcard terminal	0,8784	0,5354
Desktop Client	0,4763	0,1504
Computer	0,4784	0,4827
Fileshare	0,7697	0,3245
Browser	0,7875	0,2724
Backup/restore	0,8242	0,8071
Network Printer	0,8408	0,7530
Conventus	0,7609	0,8041
Network (Communication)	0,6194	0,4615
Patch	0,8563	0,3366
SMS Passcode	0,9598	0,7656
Sikker Mail	0,8550	0,6820
Internal Sales	0,3394	0,3223
Monitoring	0,5411	0,2754
Storage	0,6683	0,3047
Average	0,6177	0,5037

K | Unit Test: Pre-process Email

```
1  [TestClass]
2  public class PreprocessorUnitTest
3  {
4      [TestMethod]
5      public void TestingCustomPreprocessing()
6      {
7          // Arrange
8          string lowering = "TESTINGALLCAPSLOCK";
9          string closing = "best regards bob, with a lot of text after a closing";
10         string mmf = "Mail Message From:";
11         string example = "This is an example test testing the pre-processing";
12
13         string correctLowering = "testingallcapslock";
14         string correctClosing = "";
15         string correctMmf = "";
16         string correctExample = "exampl test test pre-process";
17
18         Preprocessor preprocessor = Preprocessor.GetInstance();
19
20         // Act
21         string testLowering = preprocessor.PreProcess(lowering);
22         string testClosing = preprocessor.PreProcess(closing);
23         string testMmf = preprocessor.PreProcess(mmf);
24         string testExample = preprocessor.PreProcess(example);
25
26         //Asserts
27         Assert.AreEqual(correctLowering, testLowering);
28         Assert.AreEqual(correctClosing, testClosing);
29         Assert.AreEqual(correctMmf, testMmf);
30         Assert.AreEqual(correctExample, testExample);
31     }
32 }
```

Listing 12: The *TestingCustomPreprocessing* unit test in **PreprocessorUnitTest**