Bachelor Degree Project

Predicting SNI Codes from Company Descriptions

- A Machine Learning Solution

Authors: Erik Lindholm, Jonas Nilsson
Supervisor: Johan Hagelbäck
Semester: VT 2023
Subject: Computer Science

Abstract

This study aims to develop an automated solution for assigning area of industry codes to businesses based on the contents of their business descriptions. The Swedish *standard industrial classification* (SNI) is a system used by Statistics Sweden (SCB) for categorizing businesses for their statistics reports. Assignment of SNI codes has so far been done manually by the person registering a new company, but this is a far from optimal solution. Some of the 88 main group areas of industry are hard to tell apart from one another, and this often leads to incorrect assignments. Our approach to this problem was to train a machine learning model using the *Naive Bayes* and *SVM* classifier algorithms and conduct an experiment. In 2019, Dahlqvist and Strandlund had attempted this and reached an accuracy score of 52 percent by use of the *gradient boosting* classifier, but this was considered too low for real-world implementation. Our main goal was to achieve a higher accuracy than that of Dahlqvist and Strandlund, which we eventually succeeded in - our best-performing SVM model reached a score of 60.11 percent. Similarly to Dahlqvist and Strandlund, we concluded that the low quality of the dataset was the main obstacle for achieving higher scores. The dataset we used was severely imbalanced, and much time was spent on investigating and applying *oversampling* and *undersampling* as strategies for mitigating this problem. However, we found during the testing phase that none of these strategies had any positive effect on the accuracy scores.

Keywords: *Machine learning, text classification, SNI, Naive Bayes, SVM, oversampling, undersampling.*

Contents

1 Introduction

This chapter will provide a brief introduction to the problem this thesis aims to solve.

SNI is a standard used in Sweden and stands for *Svensk Näringsgrensindelning* (which roughly translates to *Swedish industry classification).* In the SNI system, a five-digit code is assigned to a new business upon registration. The purpose of the SNI code is to enable easy categorization of companies based on which area of industry they belong to, as well as by their more specific roles within their areas of industry. Thus, the SNI code is important for tracking statistics and can help gather information on similar companies in the same industry [11]. Statistics Sweden (SCB) rely on the SNI system when producing reports on Sweden's economic and industrial growth.

Traditionally, the responsibility for selecting the right SNI code has been put on the person registering a new business. This is problematic since many of the categories are quite similar, and as a consequence, incorrect assignments due to the human factor are common. Assigning the correct SNI code upon registration is of high importance, since incorrect assignments will lead to inaccurate statistics reports.

In this thesis, we attempt to solve this problem by developing a machine learning model capable of automatically assigning an SNI number from the contents of a company description. This is done in the form of an experiment, where the effects of using different input variables on the trained and tested model are documented and analyzed. To provide context, we discuss the SNI system in further detail, and explain the technologies and challenges involved in developing an automatic solution. In the final chapters, the results of the experiment are presented along with our subsequent analysis and conclusions.

1.1 Background

Statistics and the subfield of *business statistics* is the application area of this thesis project, as we aim to solve a real-world problem for Fortnox AB: automatic assignment of SNI codes based on company descriptions. An effective solution to this problem would lead to fewer incorrect assignments, resulting in more accurate statistics reports. It would also do away with the

costs involved with the manual assignment of SNI codes, as well as make the process of starting a new business more streamlined.

The research area of this thesis project is *computer science*, and more specifically, the subfield of *Artificial intelligence.* Artificial intelligence is the field of study devoted to the development of systems and machines capable of exhibiting human-like intelligence. Within Artificial intelligence, *Machine learning* is the subfield that we expect our thesis to be most focused on. This is the field that focuses on the learning process, and how to use data to train an AI system - commonly referred to as a *model* - for a specific task.

The real-world value of our implementation will come from its ability to interpret the company description of a business and make an informed decision about its SNI code classification. Consequently, we expect that the system's understanding of human language may be a central challenge. For this reason, *Natural language processing* (NLP) is another AI subfield that is likely to be relevant. NLP is the field devoted to enabling computers to understand, interpret and generate human language.

1.2 Related work

Dahlqvist and Strandlund used Natural language processing and machine learning to aim for automation of being able to predict a Swedish business from their company description and assign the correct SNI code [1]. With the use of classification methods, they analyzed the business description to create a model that will place the industry in the area they belong to. Dahlqvist and Strandlund's thesis project was done in collaboration with Statistics Sweden, and the goal was to go from manual labor to using their machine learning model in their day-to-day production.

Unfortunately, they were unable to achieve an accuracy higher than 52 percent, which was considered too low to implement their solution in day-to-day production. Our approach will make use of similar methods for training and testing our models, but we will also explore other options available in order to improve the prediction rate. Hopefully, we will manage to create a model that can be implemented in a production environment.

Gao, He and Chen claim that with the rapid development of machine learning techniques, textual analysis methods can be of great use for finance research [2]. They started their research by first exploring the most common and recently developed techniques for text-based classifications. They then

proceeded to implement and compare the results of these techniques against the U.S. and Chinese markets. The goal was to achieve an effective way of classifying a company's area of industry.

In Gao, He and Chen's research, a combination of the word embedding scheme *latent semantic indexing* (LSI) together with *k-means clustering* gave a result that is comparable to the standard *global industry classification standard* (GICS). GICS is a tool that helps to define global industries and classifying securities by industry. It also assigns a company a code consisting of eight digits, making it similar to the Swedish SNI system. Similarly to this study, we will explore different techniques used for text-based classification, but we will specifically use the SNI as our base measure.

According to Kim, Kang, Bae and Jeon, the rapid changes in the current economy may force a company to emerge, enter or exit an industry much faster than we have seen before [3]. With this fast pace, they deem the traditional industry classifications to not be optimal and that they have certain limitations. Thus they present an alternative method for improvements to the existing industry classifications. This includes applying a text mining technique used to resolve dimensionality problems in high-dimensional texts. Using machine learning, they managed to avoid the so-called "curse of dimensionality". They claim that this approach can help improve existing text-based classification performance in predicting industry. They also show results that their approach is better than existing industry classification systems.

In order to enhance their proposed method, they deem it necessary to analyze unstructured data in financial reports. Similarly to Kim, Kang, Bae and Jeon's study, we will work with text-based classifications to determine the company's industry type from their description.

1.3 Problem formulation

We intend to use the bachelor's thesis *Predicting the Area of Industry* by Dahlqvist and Strandlund as a starting point of our thesis [1]. Our models will be trained and tested on a much larger dataset than theirs, consisting of 615 163 company descriptions with assigned SNI codes. Dahlqvist and Strandlund's best model was able to correctly predict 52 percent of the companies' area of industry. This score was considered too low to justify an implementation of an automatic system based on their model.

1.3.1 Research questions

Research question	Description
RQ1	Which of the two classification methods - Multinomial Naive Bayes and SVM - show the most potential in the context of assigning SNI codes based on company descriptions?
RQ2	What is the effect on accuracy and f1-score of applying over- and undersampling on the imbalanced dataset?
RQ3	Can an accuracy higher than 52 percent be achieved, and what can be identified as the biggest obstacle in reaching high scores?

1.4 Motivation

The end goal of this thesis project will be to develop a model that can reach a higher score than that of Dahlqvist and Strandlund's best-performing model. Ideally, the accuracy of our model will also be high enough for Statistics Sweden to be able to use it for an automatic system. Fortnox AB has expressed a strong interest in a model that could be used as the basis for such a solution.

However, even if we should fail to achieve the goal of developing a model that can be used by Statistics Sweden, our thesis will still have a scientific value. Our findings may be of help to future developers attempting to solve the same problem, or one of similar nature.

At its core, SNI number assignment is a very straightforward classification task. As such, a machine learning-trained classification model seems like an ideal solution. One of the main challenges involved, however, is the very limited amount of data provided in a company description. This is why Natural language processing and other preprocessing methods play a very important role in the project. In order to achieve a high accuracy, the text data must be processed so that the model can fully make use of every single word that carries meaning.

1.5 Results

Our best-performing Naive Bayes and SVM models were able to reach an accuracy of 56.13 percent and 60.11 percent respectively. This means that we succeeded in our goal to reach a higher score than that of Dahlqvist and Strandlund, but our results are probably still too low to justify a real world implementation.

However, one should take into account that Dahlqvist and Strandlund limited themselves to the 30 most frequently occurring classes while we only excluded classes with less than ten samples, which was only three classes with them having 3, 1, 1 occurrences. The amount of classes included has a big effect on accuracy scores, making an eight percent increase a bigger improvement than it might seem. Excluding classes can result in a boost in accuracy, but it comes at the expense of the quality of the solution and may drastically reduce its usefulness for real-world implementation. For this reason, we wanted to include as many classes as possible.

We believe that one of the success factors was that we had access to a larger dataset than that of Dahlqvist and Strandlund - 615 163 samples versus 7846. However, just as Dahlqvist and Strandlund considered their poor quality data to be their main obstacle towards reaching higher scores, our experiment was also hampered by a less than optimal dataset. Our dataset was severely imbalanced, with 62 710 samples in the largest class as opposed to less than ten in the smallest. We tried using oversampling and undersampling to mitigate this problem, but in the end neither of these techniques had any positive effect on our results.

1.6 Scope/Limitation

When choosing what programming language to use, we considered both Python, Java, and R, as these are all very popular languages for machine learning applications. While we were both somewhat more familiar with Java, Python seemed like the better choice because of its large number of AI libraries. For our IDE we chose Jupyter Notebook.

With the huge variety of machine learning algorithms available, we had to limit ourselves to those we believed best suited for our particular problem. The Naive Bayes algorithm has a track record of high effectiveness for text classification - its relatively low level of complexity and high speed would allow for quick training and testing of new models [6]. The SVM algorithm

was specifically suggested by Dahlqvist and Strandlund as a classifier worth exploring further in the context of SNI code predictions [1, p. 10].

Early on we made the decision to focus on the first two digits of the SNI code - which represent the 88 main areas of industry - rather than making use of the full five-digit code and assigning sub-categories as well. We decided that we could make an attempt at building an hierarchical system for assigning full SNI codes only if we reached a very high accuracy with the two digit-system.

We considered using TensorFlow, which has a reputation for being a good end-to-end platform for producing AI solutions to real world problems. However, developing our application within the context of a large framework would have required us to spend much of our time learning how to use the framework effectively. For the same reason, we rejected the idea of attempting to solve the task by using a neural network.

Instead, we settled upon an iterative approach: we would start out small with developing a Jupyter Notebook application that wouldn't be much more complex than our past machine learning projects. We would then continuously experiment with Natural language processing through the library NLTK, Scikit-learn and other preprocessing methods.

1.7 Target group

Our implementation will be of use for Statistics Sweden, as well as for both existing and new companies. Making the step of picking the correct SNI number automatic with our implementation, we prevent mistakes from happening. This will also target people working with machine learning that will be interested in good practice regarding similar data structure works in the future.

1.8 Outline

In the following chapters, Chapter 2 covers the picked method we chose to use during this project, while also going over how we took ethical decisions into account during the project. It also covers how we worked to make our project reliable and valid. Chapter 3 will be an explanation of the theoretical background and also cover our reasoning of why there exists a knowledge gap and how we intend to work with it. The implementation will be covered in Chapter 4 going over all the realizations and all the steps taken to reach our

results. The results will be discussed in Chapter 5 where we will present all the raw results. In Chapter 6 we analyzed the results and covered what conclusions we can take from our experiments. Chapter 7 covers the discussions over the project's findings. Lastly, to finish it up we have Chapter 8 where we discussed our conclusions and what can be done for future work for further improvements on the subject.

2 Method

We decided that the best fitting methodology for this project was to do it as an *experiment*, as finding which model would be the most suitable would be an iterative process of training and testing models on different settings.

This goes hand-in-hand with the concepts of *independent variables* and *dependent variables* commonly used in experiments.

2.1 Research Project

This project was initiated by the company Fortnox AB. They are looking for a solution to automate the process of assigning correct SNI numbers based on only a company's description. In order to automate this process we will be testing different machine learning algorithms. To conduct the test we decided that we will follow the methodology of experimentation. Using experimentation, we aim to gather the data needed to answer our research questions stated in 1.3.1.

2.2 Research Methods

In order to solve the research problem, we conducted several experiments on the data set that we received from Fortnox AB. Since machine learning is a field that is heavily reliant on empirical evaluation and experimentation, we decided that an experimentation methodology was the most appropriate way to reach our solution. Finding which model is the most suitable would be an iterative process of training and testing with different settings.

The experiment methodology is good practice to use when you need to systematically evaluate and compare different machine learning techniques in a controlled environment. The concepts of *independent variables* and *dependent variables* are commonly used in experiments. In our case the dependent variables would be our performance metrics - accuracy and f1-score - and the independent variables would be our input features. Some of the input features would be deciding what value for the hyperparameters inside our ml classifiers, whether or not we will be using any over/undersampling methods and selecting value for the random seed that will be used. In machine learning experiments, all known independent variables are usually set to a default value, and for each round of experiment only one variable is changed in order to observe what impact it has on the dependent variable [14]. Per definition, experiments require that one

systematically changes one or more variables and examine their effects on the dependent variable. Consequently, many runs of experiments would have to be carried out to determine the conditions for achieving a high result that could also be considered valid and reliable [15].

When conducting each of our experiments, there were several steps of preparing the data that had to be done before it would be used to train and test the machine learning model. In each of these steps there were also many different variables that needed to be tested. The testing of different variables was conducted in an iterative manner where we started with the preprocessing of the data.

2.3 Reliability and Validity

After reaching an accuracy score we believed would be the highest possible within our timeframe, we would proceed to thoroughly test the code and model to ensure a high reliability. To make our tests more reproducible we decided to use a consistent *random seed value*. Using the same seed value across the code would guarantee that the same sequence of random numbers would be used every time the code was executed. We would also run the code on both of our computers to ensure that the accuracy remained consistent in different environments.

Another effort to ensure reliability that we decided upon would be to double-check that we had documented which exact versions of Python, Jupyter Notebook and imported libraries that were used on the final model. We will make sure that the libraries being used throughout the project are from trustworthy sources with high reputation. By relying on said libraries rather than implementing their functionality from scratch, we aim to improve the validity of our implementation by reducing the likelihood of errors in the code.

The data provided was not very balanced, as some of the SNI main categories were far more represented than others. This was a technical challenge that we identified early on, and that we concluded could make the trained model biased towards certain categories and could negatively affect its accuracy. We addressed this by experimenting with different methods for mitigating the effects of unbalanced data, such as *oversampling* and *undersampling*. We also understood that this would be an issue that we would have to factor in when evaluating the validity of our results.

2.4 Ethical Considerations

It is difficult to envision that the outcome of our experiment could cause any direct or indirect harm. As mentioned above, the current system for SNI number assignments puts full responsibility on the person starting the new business. This means that no one at Statistics Sweden or anywhere else is likely to lose their job when an automatic solution is implemented. Neither does it seem likely that our very task-specific machine learning model would contribute to any of the doomsday scenarios that are often brought up when discussing the risks associated with AI technology.

The data used to train the model was provided by Fortnox AB. Whether the included businesses have given their consent to letting their business description and SNI number be used as machine learning data should be more Statistics Sweden's responsibility than ours. However, considering that SNI numbers and business descriptions are publicly available and could hardly be considered sensitive information, the aspect of consent doesn't stand out as much of an ethical issue either way.

3 Theoretical Background

3.1 Machine learning

Machine learning is a subfield of artificial intelligence. It is devoted to exploring processes and methods for enabling computers to improve their ability at a certain task by learning from training data. This project is an example of *supervised machine learning*, where the application is provided with training data in the form of a set of input observations, along with the correct associated classes.

The goal is for the application to associate parameter values in the input data with the different classes. After being trained, the performance of the application - which is referred to as a model - is evaluated against a set of testing data where the correct classes are not provided [4, p. 59].

3.2 Natural language processing (NLP)

In the 1950s, the field of *natural language processing* emerged with the purpose of serving as a bridge between artificial intelligence and linguistics, making it a subfield of AI and linguistics. NLP techniques enable computers to understand, interpret and generate human language. NLP can be used in

the context of many different tasks, including *text classification* (which is described below), language generation, machine translation, *named entity recognition*, *sentiment analysis*, *speech recognition* and *text summarization*.

NLP techniques are applied when there is a need to analyze text data. Our implementation of NLP will be used in the preprocessing step of the data, this step is only required to run once. That is to say the only step in our experiment that will make use of NLP is the step where we prepare and analyze the data. In order to analyze raw text data, inside NLP there is a large variety of tasks you can choose from. The tasks we selected to work with in order to preprocess our data was:

1. **Sentence boundary detection**: In most languages one can tell where dividing a string and making it into a sentence component by the punctuations. One thing that complicates this task is when it encounters abbreviations or titles.

2. **Tokenization**: This task separates full texts into words and identifies individual tokens. It's important to keep commas and punctuations as tokens since they serve as useful information on sentence boundaries. In some cases, it might be necessary to tokenize multiple words such as *rock 'n' roll* into a single token. To solve this, tokenization is commonly used with *named entity recognition* [4].

3. **Stemming and lemmatization**: Transforms the words into their root value by removing suffixes, normalizing the words into a simple common base form. Stemming and lemmatization do have some differences. Stemming is usually a process that cuts off the ends of the words in a very crude, heuristic way. Lemmatization methods are done by doing a vocabulary and morphological analysis of the words, with the goal of returning the dictionary's base version of that word [17]. In some non-English speaking countries where the language is very synthetic, there are cases where the whole sentence can be replaced by a compound word [4].

4. **Stop word removal**: Is a task that aims to remove words that don't add much information to the text but which are frequently used. Some examples of this are words such as "the", "a" and similar.

3.3 Text classification

Text classification is a technique that makes use of machine learning and in some cases makes use of NLP to analyze and structure text, so that information can be gathered from it. A text classifier is trained and tested through machine learning. It is then used to classify input data - in the form of open-ended text - into one of a predefined set of classes.

A popular example of text classification is sentiment analysis, which is used to determine whether the author of a text has positive or negative feelings towards a certain object. For example, a web shop can use sentiment analysis on customer reviews to determine how the customers feel about their products [4].

3.4 Bag of words

A *bag-of-words* refers to a representation of a text document where the included words' positions in the original document have been ignored. Instead, only the frequency of each word is kept, which makes the bag-of-words representation suitable for quantitative analysis [4, p. 60]. Individual words are given full focus, while their order within a sentence or a longer text is considered unimportant.

The words of the source text are mapped to a *vector*. Gao, He and Chen uses the following example to describe how the vector mapping works:

> "For example, we have two sentences: 'Here are a white cat and a black cat' and 'Here is a dog'. The set of words used here are Here, are, is, a, white, black, cat, dog. The bag of words vector representation of these two sentences are [1,1,0,2,1,1,2,0] and [1,0,1,1,0,0,0,1]." [2, p. 2].

3.5 Oversampling and undersampling

An imbalanced dataset suffers from a significant skew in the distribution of classes, such as 1:100 or 1:1000 examples in the minority class to the majority class. This can have a negative impact on the training and testing of a machine learning model. *Oversampling* and *undersampling* are two approaches to deal with this issue and mitigate the effects of the imbalance [9].

The basic idea of oversampling is to generate new samples in the minority classes. There are several different methods for doing this. Naive random oversampling generates new samples from existing ones, which are picked at random. Two other popular algorithms are *Synthetic Minority Oversampling Technique* (SMOTE) and *Adaptive Synthetic* (ADASYN). Rather than duplicating original samples, SMOTE and ADASYN use interpolation to generate new synthetic samples.

Undersampling instead attempts to balance the dataset by randomly deleting samples in the majority classes. Like with oversampling, there are a number of different algorithms that can be used. A *prototype generation* algorithm creates a new set from the targeted majority class, where the amount of samples are reduced and those that are included are generated from the original data. *Prototype selection* algorithms, on the other hand, select samples from the original set and use them to populate a new reduced set.

Oversampling can put a model at risk for overfitting, and the backside of undersampling is that it can result in loss of information that could be valuable to the model [8].

3.6 Naive Bayes

The *Naive Bayes algorithm* is a common and fairly basic classification algorithm based on *Bayes' theorem*. Bayes' theorem uses prior knowledge of conditions that may be related to an event to determine the probability of it. It is stated mathematically with the following equation:

$$P\,(A \mid B) \;=\; \frac{P(B \mid A) * (P \mid A)}{P\,(B)}$$

Equation 3.1: Bayes' theorem [6].

In equation 3.1, "P (A | B)" can be read as "the probability that A occurs given B". Bayes' theorem only takes one attribute into account when calculating the probability of an item belonging to a specific category. However, a Naive Bayes classifier combines several attribute inputs to determine the probability, as this is what is generally needed for real-life use cases.

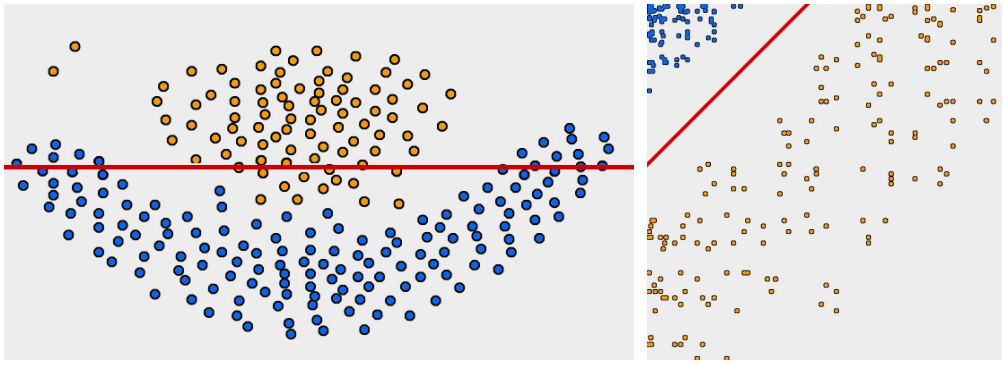
The reason the algorithm is called naive is because it makes the assumption that the attributes are independent of one another. This is most oftenly

wrongful, as there tend to be relationships between attributes. This means that a Naive Bayes classifier can never be assumed to calculate an item's *actual probability* of belonging to a class.
Despite this inherent problem, the algorithm also has several strengths such as high speed and scalability, and Naive Bayes classifiers have turned out to be highly useful in practice. A classifier uses Bayes' theorem to calculate probabilities for a number of classes, and the one with the highest score is then selected. This has proven to be a good-enough solution to a great number of real-world classification tasks [6].

3.7 SVM

Support-Vector Machine (SVM) is a classifier that relies on kernel methods, which are all developments of the more basic *Linear Kernel Classifier*. The Linear Kernel Classifier first calculates a center point in space for each category. A category's center point is based on the average of each attribute value for every included example. A new example is then classified based on which of the categories' center points it is closest to - usually this is measured by Euclidean distance.

To produce more accurate results, more complex versions need to transform the data. To do this they make use of the Kernel Trick, which requires measurements between *dot-products* rather than between coordinate vectors (which are usually used for measurements in Euclidean spaces).



Figures 3.1 and 3.2: A linear classifier will give inaccurate results if the problem isn't linearly separable. Kernel classifiers address this by applying data transformation. Figure 3.1 (left) shows the Flame dataset, and how it is impossible to place a line on it so that the samples in the two classes (red and blue) are completely separated. Figure 3.2 (right) shows the Flame dataset after every x and y value in the dataset has been squared. The problem has become linearly separable from the data transformation. Images reworked from [7].

A dot-product is a single numerical value calculated as the sum of the products between each of the values in the first vector and their corresponding values in the second vector.

$$dot(v0,\ v1)\ =\ v0_0\ *\ v1_0\ +\ v0_1\ *\ v1_1\ +\ \ldots\ +\ v0_n\ *\ v1_n$$

Equation 3.2: How to calculate dot product [7].

The Support-Vector Machine algorithm uses the averages of each category to determine dividing lines between them, and new examples are then categorized based on their position. A problem with this approach, however, is that faraway examples will affect the placement of the lines, but only the examples closest to the dividing lines will actually be relevant to determining their exact placement.

To mitigate this problem, the algorithm makes use of a *Support-Vector Machine* which finds the line that is as far away as possible from each of the categories - the so-called *maximum-margin hyperplane*. The maximum-margin hyperplane can be found by drawing imaginary lines between each of the examples in a category, using the outermost lines to

define a polygon - called the *convex hull* - and then placing the hyperplane exactly between the convex hulls of the two categories.

Defining the hyperplane actually doesn't require all examples to be taken into account, but only those closest for each category. These are referred to as the Support Vectors [7].

4 Research project – Implementation

This chapter contains a thorough explanation of all the steps involved in training and testing a model in the context of this experiment. Starting from how we first manage the data sets and the way we process the data in order for us to conduct tests with our different algorithms and how we have fitted our models.

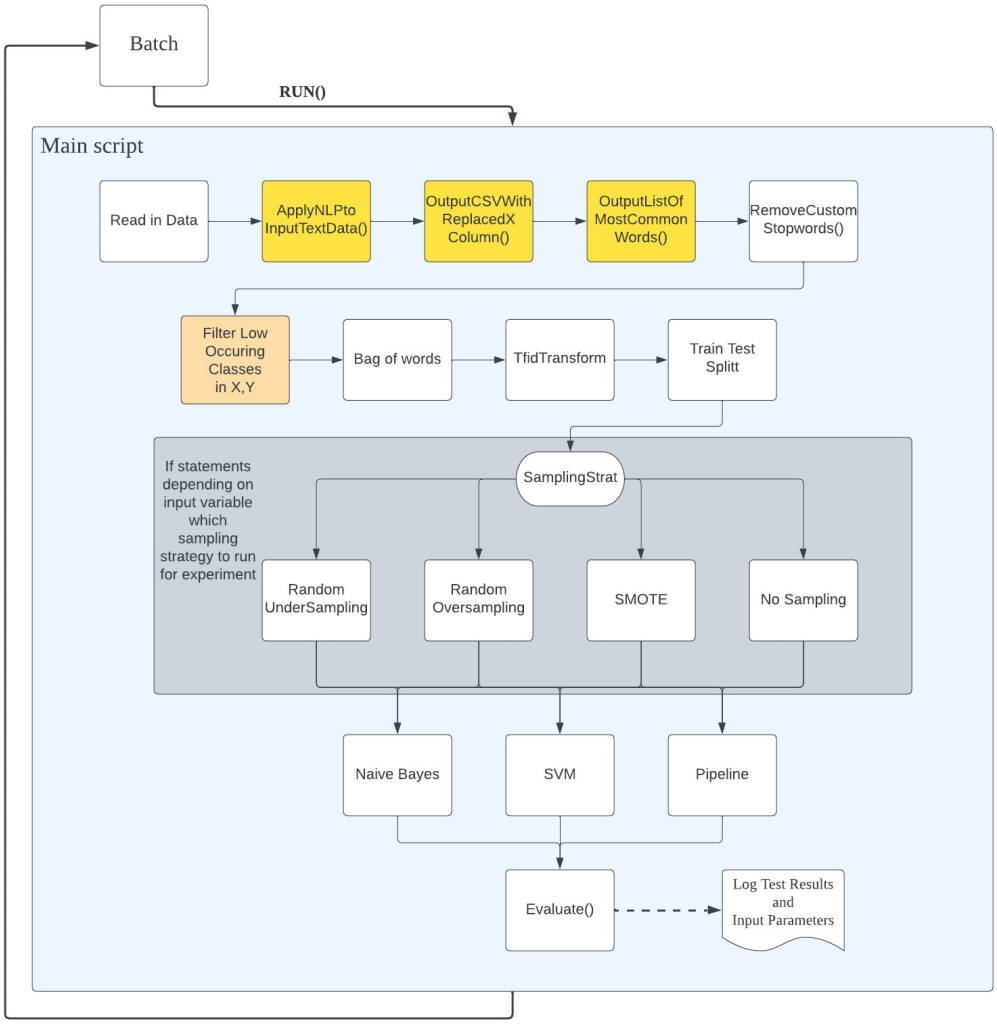


Figure 4.1: Shows the process of an experiment in our ML solution. The main script will be run from our batch script, where we can run several tests. Steps 2-4 which have the background color yellow is to show they are only needed to run during the very first experiment to prepare the dataset. The orange-colored step is only done if there are any classes with 10 or fewer instances. At the end of Evaluate() we will also document the result from each test in an excel file and a text file.

We will also cover the different tweaks that have been made for all the used methods during the project, which were tested to aim for a better result. This result will be covered later in Chapter 5, where the results of each test with different tweaks will be presented, followed by Chapter 6 where we will analyze the results.

4.1 Batch Execution

The application for training and testing models use two Jupyter Notebook scripts: the *main script* and the *batch execution script*.

The batch execution script allows the main script to be run several times with different input arguments. Input arguments are defined in the *args* variable, and the main script is executed by calling the *Run()* function.

Every time the main script is run, the results and input arguments are saved as a JSON file in the *test-logs* folder. For easy overview, the folder also contains an Excel file where the name, date and accuracy scores of the test are appended automatically.

4.2 Read in Data

At the very start of the process, the popular *pandas* library is used to read and parse the data from the CSV files. Once the data has been loaded into a pandas DataFrame object, the contents of the DataFrame are checked for NaN values. This is done by a simple method call: *df.fillna(' ').values* and it replaces any NaN values with a space.

Before moving on to process the company descriptions, we extract the two relevant columns from the DataFrame - the company descriptions and the SNI numbers - and store them in variables. If the parameter *data_limit* has been set to a number lower than the amount of rows in the DataFrame, the columns are also cropped (this is to facilitate faster testing and training on a smaller dataset).

4.3 ApplyNLPtoInputTextData

In this step, a number of NLP operations are performed on the company descriptions in the dataset to make them better suited for machine learning.

The first operation performed is *tokenization*. This is the process of breaking up a string of text into semantically useful units like words or sentences. In

our case, the company description is tokenized into individual words: “We build houses and make money” becomes “We”, “build”, “houses”, “and”, “make”, “money”.

The tokenization is followed by some simple operations such as converting the tokens to lowercase letters and removing numbers and special characters. More complex operations then follow - *stemming* and *lemmatization*, which are two different approaches for converting words to their meaningful base forms (for example, “going” becomes “go”).

The company description is also checked and filtered against NLTK’s built-in list of so-called *stopwords*. Stopwords are high-frequency words that add very little semantic meaning to a sentence, for example *at, for, is, which, to* (*vid, för, är, vilken, till* in Swedish). Aside from the filtering of general stopwords, we have also added our own solution for filtering out stopwords that are more specific to our data - this will be described in further detail below.

4.4 OutputCSVWithReplacedXColumn

Applying the NLP operations to the data takes a considerable amount of time, and it is actually not necessary to do this every time the code is run and a new model is trained. If the data and the NLP operations remain the same, the dataset can be saved after processing and then loaded the next time the application is executed.

This is what the OutputCSVWithReplacedXColumn function is used for - it takes the original dataset and replaces the column containing the company descriptions with the same column from the processed dataset. The dataset is then saved as a new CSV file, which can be loaded on start-up instead of the original data.

4.5 OutputListOfMostCommonWords

OutputListOfMostCommonWords is not meant to be run when training and testing the model - it is a function that was used in the process of creating a list of custom stopwords. The function counts the number of occurrences of words in the dataset and orders them in a descending list. By manually going through the list and deciding which of the words should be filtered out, a list of custom stopwords can be created.

This was how we created our lists of custom stopwords, where for example a word like “company” was considered to have too little value to keep, but “publishing” was kept as it indicates a specific business category. A hashtag prefix means the word is considered valuable and should be kept.

4.6 RemoveCustomStopwords

This is the function that is responsible for actually filtering out the words included in the custom stopwords list. It opens the stopwords text file, then iterates through every company description in the NLP-processed dataset. Every token matching a non-hashtag-prefixed word in the custom stopwords list is then removed.

Removing custom stopwords is a much faster process than applying the NLP operations, so it is performed every time the application is executed.

4.7 Filter low-occurring classes

From the very start, we knew that one of the major challenges of our experiment would be the extremely imbalanced dataset. While there were tens of thousands of instances in some SNI number categories, there were less than ten in some. After much discussion on how to address this problem, we came to the conclusion that we had to exclude the categories with extremely low numbers of instances. Allowing these underrepresented categories to remain would cause problems in later steps.

We set the limit to a minimum of 10 instances. A method from the package called *numpyindexed* is responsible for filtering out the minority classes.

4.8 Bag of words

In the context of our machine learning application, a word’s number of occurrences is much more relevant than its position within the text. So, to make full use of the processed company descriptions as input data for our model, it is necessary to convert them to bag-of-words representations.

In this step, we apply bag-of-words conversion to our raw input data with the *CountVectorizer()* method. This method is imported from the *sklearn.feature_extraxtion.text* library.

4.9 Convert from occurrences to frequencies

The class *TfidfTranformer* is imported from the same library as the bag-of-words method used in the previous step. By using this class, the importance of each word in the input data is measured.

Words that appear often will generally be valued higher than words that have a low occurrence. However, a high-frequency word will get a lower value if it appears frequently in a great number of categories, as opposed to if it is only recurring in a few categories. there will be some words with high frequency across many descriptions that will result in them having lower values.

For our use case, we apply two TfidfTransformer methods - *fit()* and *transform()* - on our *input* data (the bag-of-words converted company descriptions). An informative scheme on the word frequencies in the input data - where every word now has a weight value assigned to it - is returned and assigned to the input variable.

4.10 Train and Test Data Split

In order for us to follow good practices it is necessary to split the data into a training set and a testing set so that the model can first be trained on one set of data and then tested against another. We opted to make use of the *sklearn.model_selection* method called *train_test_split*, as this method provides you with an easy way to adjust the proportions of the splits.

During this part of the experiment we did many variations of parameters, like trying out different split proportions, and checking whether using a *stratify* method yielded a different result. The use of stratified parameters can give a better representation of each class in our dataset.

4.11 Oversampling

In order to make our dataset more balanced as the datasets classes had a large difference in occurring instances.

As mentioned above in Chapter 4.6, the dataset initially had classes with as low as one instance, which caused a lot of difficulties. Even though the least represented classes are filtered out by the *numpyindexed* method call, the remaining dataset is still severely imbalanced. So in this step, oversampling is applied to mitigate the effects of the imbalance by expanding the minority classes.

We have been running experiments using both *random oversampling (ROS)* and *SMOTE* in order to see what works better with our dataset. During our experiments with oversampling, the *sampling_strategy* that is most commonly used is the default setting *all*, which is the one we started with. This strategy iterates over all classes and expands them to the size of the largest class. For each experiment we used one of the sampling methods, so which one was used is clearly stated as an input variable. The different parameters one can choose from is *minority, not minority, not majority.* For each of these parameters, rounds of tests will be conducted. Another parameter that both *ROS* and *SMOTE* have in common is *random_state* which is used to control the randomization of the algorithms.

When working with *SMOTE* we also have a parameter called *k_neighbors* and it is used to define the nearest neighbor in order to define a neighborhood for the synthetic samples [12]. During our experiments with *SMOTE* we decided to experiment with the values 1-10 for our *k_neighbor* variables.

4.12 Undersampling

As described in chapter 3.6, undersampling takes the opposite approach to dealing with the imbalanced dataset by removing instances of the larger classes to make them more similar in size to the smaller ones. We have conducted some experiments using the *Random Undersampling (RUS)* method in order to make this happen. When using undersampling we have mostly relied on the default setting for the *sampling_strategy* parameter, which reduces all classes into the same size as the smallest ones. However, it is inevitable that applying undersampling like this results in the removal of a lot of data that would have been valuable to the model.

4.13 Classification algorithms

Two different classification algorithms have been selected for training and testing models within the experiment.

As mentioned in Chapter 3.6, *Naive Bayes* has proven to be a very good algorithm when it comes to classification. Our implementation uses the *MultinominalNB* classifier from the *sklearn.naive_bayes* package. The parameter *alpha* is set to different values in different tests, to see which value produces the best results on our dataset.

The second algorithm that we have decided to use is the *Support-Vector Machine* (SVM) algorithm and inside of the SVM we decided to make use of the *LinearSVC*. SVM offers a greater number of adjustable parameters than Naive Bayes - the ones we experiment with different settings for are *C* and *class_weight*. *C* is a regularization parameter where the strength is proportionally inversed to C and the number for this parameter is required to be a positive number. Where a lower number means that more errors during training is allowed than if you use a large number. *Class_weight* takes the values of y and adjusts the weight automatically according to the class frequencies [13].

4.14 Evaluation

During our evaluation function we will start with training the model against our training set, by using the method *model.fit(X_train, y_train)* we can use this for all the models in order to train them. When the training step is done we will run a prediction method with the parameters of our input data from the test split to be able to plot how the model's predictions are. When the prediction method is complete we will be running a method to calculate the accuracy of our model's predictions, this will be done with the use of the *accuracy_score()* method with the parameters, *test labels* and the prediction that was made in the earlier step. When we have calculated our accuracy we will create a confusion matrix with the use of the *sklearn.metrics* library to import *confusion_matrix()*. For the confusion matrix, we will provide the test labels and the predictions as parameters. When this confusion matrix is done we will print out a classification report, by using the library *sklearn.metrics* we can make use of the *classification report()*. Similar to the previous step the parameters will be the same, *test labels* and the prediction and it will print out the result of the test. When we have printed out all the results from the first part of our *evaluate* function we will continue to the next step which is to do a procedure called *5-fold cross-validation*.

The purpose of *cross-validation* is to estimate how well a machine-learning model can handle unseen data. It is a procedure that uses a limited amount of samples in order to see how well the trained model can predict data that it has never seen during its training process [10]. By using the sklearn library we can import a method that is called *cross_val_predict()*, where we will provide four parameters. The first parameter will be for the model we are using, the second parameter will be where we put our input data, the third parameter

will have the labels data, and lastly, the fourth parameter will decide how many folds it will be during this process. When this *5-fold* procedure is completed we will make use of similar methods as in the previous step. First, we will start with calculating the accuracy with the *accuracy_score* method, followed by using the *confusion_matrix* method and lastly, we will use the *classification_report*. When this step is printed out we have gotten to the end of our *evaluate* function and can start to look at the results of the model's training.

5 Results

5.1 Defining input parameters

The batch script described in the last chapter uses a default set of input parameters defined in the default_args dictionary. Certain parameters are then altered for each test in order to evaluate their effect on the result. When a parameter setting has been concluded to be optimal or near-optimal, the default_args dictionary is then changed so that the setting that results in the highest accuracy score and f1-score will then be used by the default in all subsequent tests:

Argument	**Default Value**
name_of_data_file	“nlp_applied_data_0”
name_of_custom_stopwords_file	“custom_stopwords_1”
data_limit	900000
use_full_SNI_numbers	False
random_state	42
naive_bayes/on	True
naive_bayes/alpha	1
svm/on	True
svm/C	1
svm/class_weight	“balanced”
pipeline/on	False
split_training_and_testing_data/test_size	0.2
split_training_and_testing_data/stratify	False
under_sampling/on	False
under_sampling/sampling_strategy	“all”
filter_low_occuring_labels/on	True
random_oversampling/on	False
random_oversampling/sampling_strategy	“all”
SMOTE/on	False
SMOTE/k_neighbors	1
SMOTE/sampling_strategy	“all”
combine_under_and_oversampling/on	False
combine_under_and_oversampling/over_sampling_strategy	“minority”
combine_under_and_oversampling/under_sampling_strategy	“majority”
testing_methods/train_test_split	True
testing_methods/5-fold	True

Table 5.1: The default parameter settings defined in default_args at the start of the testing process.

Note, however, that not all of these parameters were changed during the course of the experiment. In the following chapter, we will describe the process of how we determined the final settings for all of the parameters that we evaluated.

5.2 Custom stopwords list

We had produced two lists of custom stopwords to use with the tests. *custom_stopwords_0* were the more forgiving list, while *custom_stopwords_1* would exclude more words (at the risk of excluding words that could be of value to the model). We began testing using *custom_stopwords_1*, but after conducting a number of dedicated *Naive Bayes* tests for evaluating which list performed best, we found that *custom_stopwords_0* produced a better result than both *custom_stopwords_1* and using no custom stopwords list at all. From this point and onward, we decided to make *custom_stopwords_0* the default list to use.

5.3 Split Training and Testing Data

When starting with the *train_test_split* implementation, we needed to decide on some input variables to start with. During the experiments we decided that the variables to test out were the ones called *stratify* and *test_size* parameters. Our starting variable for this was 0.2 for the *test size* while not having any *stratify* parameter enabled. After some tests were conducted and also through some discussions we finalized that the best parameters for this part were to use *stratify=y*, where y is equal to the labels and to make use of a little smaller test size so we saw the best results from having our test size set to 0.15.

5.4 Undersampling and Oversampling

Undersampling and oversampling turned out to have no positive effect on accuracy when used with neither the *Naive Bayes* nor the *SVM* classifier.

5.5 Naive Bayes

The tests using the *Naive Bayes* classification algorithm finished much faster than those using *SVM*, which led to two different approaches when conducting tests. We started the testing process by running a large number of tests with the *Naive Bayes* classification algorithm, and a much smaller number of tests using *SVM*.

The *Naive Bayes* parameter that we mainly focused on was the *alpha* value. Alpha is a *hyperparameter,* which means it is a parameter whose value controls the learning process of the model. Its purpose is to handle the issue of zero probabilities, and it is sometimes also referred to as the *smoothing parameter* or *Laplace smoothing* [16].

Our default setting for alpha was 1, and we quickly noted a trend in the test results - raising the value would produce worse accuracy scores, and lowering it would increase them. A setting of 0.02 initially gave the best result (test a25, accuracy: 54.10% f1-score: 53% in the train-test split, accuracy: 53.49% f1-score: 52% in the 5-fold cross-validation). Through further testing, we found that 0.016 and 0.017 achieved an even better result (tests a81-a82, accuracy: 56.12% and f1-score: 54% in train-test split, accuracy: 55.32% and f1-score: 54% in the 5-fold cross-validation).

We continued to try out undersampling, random oversampling and SMOTE with different settings along with alpha value 0.017. However, neither of these methods had a positive effect on the accuracy - after the tests had finished, the best-performing Naive Bayes tests were still tests a81-a82.

5.6 SVM

During our testing with the *SVM* we decided to find the most optimal value for the parameter *C*. As mentioned in Chapter 4.13, C is a *hyperparameter* used in SVM that handles regularization. This number decides how strict errors should be handled during the training process. At first we tried with values in the range from 0.1, 0.5, 1, 10, 100, 200 all the way up to 1000. During this, we could clearly see that the time needed for each test went up by 30-60 minutes for each step, at least in the hundreds numbers. While time was also taken into account we could also see a clear decrease in the scores produced from this, which made us decide to not go that high with our *C* parameter.

At this point, we also had a discussion regarding the use of *5-fold cross-validation* for our *SVM* algorithm. Time was a deciding factor for us to be able to complete the thesis project on schedule, and *SVM* is an algorithm that takes more time to run than the *Naive Bayes* algorithm. Therefore from this point on we decided to not include any more *5-fold* tests for our *SVM* experiments. Also, we had observed that the *5-fold* results were always a

small margin below those of the *train-test split* method, which made it more time efficient to just rely on the latter.

After some more testing, we found that the best looking results for our *C* parameter were between 0.16, 0.17, 0.18 and 0.19. From this, we decided to take the median value which is 0.175 which we rounded up to 0.18 (test b64 accuracy: 59.89%, f1-score: 58% in train-test split). Which gave the best result at this point in time and used it as a standard value for our *C* parameter.

When we had our *C* parameter decided, the next step in our testing would be to conduct experiments with sampling methods. The sampling method we started out with was *random oversampling*. In sampling strategies, we tried out different settings for the parameter called *sampling_strategy* in order to find out which worked best for our data. We quickly came to the conclusion that *random oversampling* led to a decrease in every test, the only times it was only a slight decrease was when only the minority got sampled or when we used the *sampling_strategy* to increase all classes with under 200 samples up to 200. With this number we could see it could reach the same result of our earlier highest achieved, if we tried to increase the sampling to higher numbers we discovered that it was decreasing again.

Running tests with *SMOTE* and undersampling also resulted in decreased accuracy scores. Even when we tried out *SVM's* own *class_weight* parameter and set it to *balanced*, there was a clear decrease in how our model performed with all of the above methods.

5.8 Final results

After adjusting the *random_state* argument, our best *Naive Bayes* model produced an accuracy score of 56.13 percent and with a f1-score of 54% (test a151). Our best *SVM* model was roughly four percent better, with an accuracy score of 60.11 percent and with the f1-score being 59% (test b124). To ensure that these results were reliable, we switched computers and trained and tested two new models using the exact same input parameters (tests a160 and b160). As expected, the scores of these models were identical with those of their counterparts b124 and a151. The characters “a” and “b” in the test names refer to which computer the test was executed on, so for example, the identifier b124 should be read as “test number 124 on computer b”.

Argument	Default Value
name_of_data_file	“nlp_applied_data_0”
name_of_custom_stopwords_file	“custom_stopwords_0”
data_limit	900000
use_full_SNI_numbers	False
random_state	12
naive_bayes/on	True
naive_bayes/alpha	0.017
svm/on	False
svm/C	1
svm/class_weight	“balanced”
pipeline/on	False
split_training_and_testing_data/test_size	0.15
split_training_and_testing_data/stratify	True
under_sampling/on	False
under_sampling/sampling_strategy	“all”
filter_low_occuring_labels/on	True
random_oversampling/on	False
random_oversampling/sampling_strategy	“all”
SMOTE/on	True
SMOTE/k_neighbors	5
SMOTE/sampling_strategy	{ "4": 200, "6": 200, "9": 200, "12": 200, "16": 200, "18": 200, "32": 200, "35": 200, "51": 200, "56": 200, "72": 200 }
combine_under_and_oversampling/on	False
combine_under_and_oversampling/over_sampling_strategy	“minority”
combine_under_and_oversampling/under_sampling_strategy	“majority”
testing_methods/train_test_split	True
testing_methods/5-fold	False

Table 5.2: Naive Bayes a151 model input parameters.

Argument	Default Value
name_of_data_file	“nlp_applied_data_0”
name_of_custom_stopwords_file	“custom_stopwords_0”
data_limit	900000
use_full_SNI_numbers	False
random_state	5
naive_bayes/on	False
naive_bayes/alpha	1
svm/on	True
svm/C	0.18
svm/class_weight	null
pipeline/on	False
split_training_and_testing_data/test_size	0.15
split_training_and_testing_data/stratify	True
under_sampling/on	False
under_sampling/sampling_strategy	“all”
filter_low_occuring_labels/on	True
random_oversampling/on	False
random_oversampling/sampling_strategy	“minority”
SMOTE/on	False
SMOTE/k_neighbors	1
SMOTE/sampling_strategy	“all”
combine_under_and_oversampling/on	False
combine_under_and_oversampling/over_sampling_strategy	“minority”
combine_under_and_oversampling/under_sampling_strategy	“majority”
testing_methods/train_test_split	True
testing_methods/5-fold	False

Table 5.3: SVM b124 model input parameters.

6 Analysis and Discussion

The final results answer **RQ1**, as SVM beat the best Naive Bayes score by four percent. With our best-performing SVM-trained model reaching an accuracy of 60.11 percent, we also succeeded in answering the first part of **RQ3**: an accuracy higher than Dahlqvist and Strandlund's 52 percent was indeed possible to achieve. However, even if it's an improvement it seems unlikely that this score will be considered high enough to implement a real-world solution with our model as the basis.

Our dataset of 615 163 company descriptions with SNI numbers was severely imbalanced. Much time was put into mitigating this problem with methods such as over- and undersampling, and excluding extreme minority classes from the training and testing data altogether. However, in the end these tactics had no positive effect on the accuracy scores. In the case of our best-performing Naive Bayes and SVM models, applying over- and undersampling only either led to lower scores or had no effect at all. So, to answer **RQ2**: applying over- and undersampling to our imbalanced dataset did not produce any improvement in the performance metrics.

This was surprising, as we had expected that these methods would play a crucial role in finding the optimal model. In comparison, altering the alpha value, the C value, the random state value, and changing the custom stopwords list all at least led to minor improvements in accuracy. At the beginning of our thesis project, we were made aware that training a model in machine learning can take a very long time for each experiment. Even with that knowledge when playing around with the SVM algorithms C parameter and using 5-fold cross-validation, it took us by surprise how much time each experiment ended up running. Increasing the C value would add hours to the execution time of each test, even when executed on our fastest computer - some tests took six to seven hours to complete. From this, we can only assume that a bigger and less imbalanced dataset would take up a considerable amount of more time.

While the problems caused by the imbalanced dataset could be solved by using a better dataset, acquiring one is easier said than done. The dataset we've used is a fair representation of how the companies in the Swedish market are actually distributed. Big hard-competition industries include a large variety of companies, while small niche industries have much fewer companies.

Another issue is more difficult to solve as it is very closely tied to the nature of the problem: the very limited amount of text data provided in a company description. How many of the final model's errors are caused by the imbalanced data? And how many are caused by it being forced to guesswork when presented with only a short sentence without any valuable words in it?

The outcome of our experiment highlights the importance of a balanced dataset when training a machine learning model, as well as the importance of providing it with enough data to actually make accurate predictions on new samples. This provides us with an answer to the second part of **RQ3**: the biggest obstacle preventing higher scores appears to be dataset imbalance combined with low-quality data.

Another approach that was considered at the beginning of the experiment was to train a model on company names rather than descriptions. This would have restricted the model to even less input data - while it would have been another experiment altogether, it's easy to see a scenario where that model would have struggled even more with making predictions based on other than guesswork.

While the research conducted by Gao, He and Chen and Kim, Kang, Bae and Jeon provided interesting and unorthodox angles on how to approach the problem, Dahlqvist and Strandlund's study was by far the most relevant. It outlined many of the challenges that turned out to be specific to the Swedish SNI code system, and after finishing our experiment, our results and subsequent analysis are largely similar to theirs. They too were hampered by a low-quality dataset in their pursuit of high accuracy scores, but unlike us they didn't actively try to mitigate the problem through over- and undersampling.

Another difference worth noting is that Dahlqvist and Strandlund's dataset was much smaller than ours - their original dataset consisted of data from 95 450 Swedish companies registered between 2008 and 2019 with at least one employee. Out of these, they decided to reduce the amount even further by using a subset of only 7846 companies which had ten or more employees, based on a claim from SCB that these were more likely to have high-quality company descriptions [1, p. 3]. This approach was never really an option for us, as our dataset didn't contain any other information than company descriptions and SNI codes.

With our results in hand, focusing as much on over- and undersampling as we did may seem like a wrongful prioritization. However, it can be argued that exploring the effects of these methods was a better choice than just leaving the obvious problem with the imbalanced data unaddressed. It should also be emphasized that even though over- and undersampling didn't have any positive effect on our results, these strategies should not be dismissed as useless. In a different context than with our specific problem and dataset, they may very well produce an increase instead.

As it turns out, Natural language processing also ended up playing a much smaller role in our experiment than we had anticipated. Some NLP methods were applied during the preprocessing of the data, such as stemming, lemmatization, tokenization and removal of general Swedish language stopwords. However, the possibilities to adjust the settings of these methods in a way that would lead to higher accuracy or f1-scores seemed very limited. Consequently, no particular focus was placed on the NLP preprocessing when carrying out our tests.

8 Conclusions and Future Work

With the increasingly sophisticated machine learning algorithms of today, creating an automated solution for the SNI code system has never seemed like an impossible task. However, the importance of high-quality data in the machine learning process shouldn't be underestimated. The massive amounts of data produced by an increasingly digitalized and connected world has been a key enabler of the last decade's rapid advancements within the fields of AI and machine learning.

In the case of the SNI code system, real world circumstances limit the availability of data as the model has to be trained on Swedish company descriptions and SNI codes. Unless you produce a dataset of made-up company descriptions for the purpose of training a model, you will have to make do with data gathered from actual registered companies. Sweden is a country with a small population - estimated to roughly ten million people in 2023 - so the amount of available company descriptions will be relatively few. Consequently, the training data available for the areas of industry that are the least represented in Sweden will be severely lacking.

Our experiment was a success in the sense that we achieved our main objective - with our top score of 60.11 percent, we reached an 8.11 percent accuracy improvement over Dahlqvist and Strandlund's best model. In our particular case, over- and undersampling turned out to have little to no effect on the end result, despite being go-to methods for countering problems with imbalanced data. Perhaps unsurprisingly, the data a model is trained and tested on may turn out to have a much larger impact on its performance than the preprocessing, algorithms and settings used when training it.

If time and resources had allowed it, it would also have been interesting to add an implementation to calculate the probabilities of the output from the SVM classifier, by using techniques such as *Platt scaling* or other similar methods. Using this may have provided us with a clearer image of the probabilities of each class, from which we could have created a top-5 ranking list. However, in order to avoid overfitting in this operation, an internal 5-fold cross validation check would have to be carried out, making it very time-consuming to run tests with such functionality.

Dahlqvist and Strandlund, who completely relied on the gradient boosting algorithm, concluded their thesis by stating that using other machine learning

algorithms such as SVM could possibly increase accuracy. They also stated that the improvement, however, would most likely not be very significant. Our results show that they were correct in both of these assumptions. Dahlqvist and Strandlund also came to the conclusion that one of the biggest issues that they had to combat during their study was the poor quality of data. They mention the fact that there is no baseline of reference on the content or structure of a business description:

> "Business descriptions containing a single word does not provide much information about the area of industry. For example, describing a business with only the word *cars* will not provide any information if cars are sold, produced or for rent, which would all yield different SNI codes. Antipole of these businesses is those who try to write everything the business might do in an infinite lifetime. Their main business might be the production of cars, but they will include the sale of cars to allow for pivoting if circumstances change [1, p. 11]."

Furthermore, Dahlqvist and Strandlund expressed their hope that Statistics Sweden's implementation of a new system for digital annual reports - DiÅR - would lead to a better structured dataset, which could be used to train a machine learning model. Since we cannot compare Dahlqvist and Strandlund's dataset to ours, we can't be certain whether our higher results can be attributed to the SVM algorithm or to a somewhat better dataset.

The fact that Dahlqvist and Strandlund managed to reach 52 percent with a much smaller dataset - 615 163 samples versus 7846 - also begs the question whether quantity or quality is the key to an optimal dataset. Our dataset consisted of only company descriptions and SNI-numbers, so we had no other data that could be used to distinguish between high and low quality samples. A dataset as large as ours, but with additional information about the companies, could perhaps allow for filtering out low-quality samples while still providing the model with enough data?

Something that should be considered, however, is that out of the 88 main groups, Dahlqvist and Strandlund chose to focus on the 30 most frequently occurring, while we only excluded the very few that had less than ten samples [1, p. 7]. The amount of categories included have a large impact on

the accuracy of a model, making the eight percent rise a much bigger improvement than the scores alone might suggest.

Whether taking the different approaches into account or not, the eight percent improvement in accuracy is highly encouraging. Today's booming interest in AI technology will most likely result in new and improved methods for text classification. Combine these with even more data gathered through DiÅR, and the next attempt at developing an automated SNI code assignment system may continue the trend towards even higher accuracy scores.

References

[1] P. Dahlqvist-Sjöberg and R. Strandlund “Predicting the Area of Industry: Using machine learning to classify SNI codes based on business descriptions, a degree project at SCB,” (June 25, 2019). [Online] Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1329995&dswid=-965. [Accessed Jan. 24, 2023].

[2] H. Gao, J. He and K. Chen “Exploring Machine Learning Techniques for Text-Based Industry Classification,” (June, 29 2020). [Online] Available at SSRN: https://ssrn.com/abstract=3640205 or https://dx.doi.org/10.2139/ssrn.3640205. [Accessed Jan. 25, 2023].

[3] D. Kim, H. Kang, K. Bae and S. Jeon “An Artificial Intelligence-Enabled Industry Classification and its Interpretation,” (March 2022). [Online] Available at emerald insight: https://www-emerald-com. [Accessed Jan. 25, 2023].

[4] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. draft, Stanford, CA, USA, 2023.

[5] P. M. Nadkarni, L. Ohno-Machado, W. W. Chapman “Natural language processing: an introduction”. (Sept. 01, 2011) [Online] Available at Oxford Academic/JAMIA: https://doi.org/10.1136/amiajnl-2011-000464. [Accessed Apr. 04, 2023].

[6] J. Hagelbäck, Linnaeus University, Växjö, Sweden. Naïve Bayes Algorithm. (Aug. 30, 2019). Accessed Apr. 5, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=ipJAtpYxlrY

[7] J. Hagelbäck, Linnaeus University, Växjö, Sweden. Applied ML - Kernel Methods and SVMs. (Feb. 14, 2018). Accessed Apr. 5, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=e3TzdioY2hI

[8] imbalanced-learn “Problem statement regarding imbalanced data sets.” [Online]. Available: https://imbalanced-learn.org/stable/introduction.html. [Accessed Apr. 6, 2023].

[9] J. Brownlee. “Random Oversampling and Undersampling for Imbalanced Classification.” (Jan. 5, 2020). [Online]. Available: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/ [Accessed Apr. 6, 2023].

[10] J. Brownlee. “A Gentle Introduction to k-fold Cross-Validation.” (May 23, 2018). [Online]. Available: https://machinelearningmastery.com/k-fold-cross-validation/. [Accessed Apr. 19, 2023]

[11] Statiska Centralbyrån “Sökning efter SNI-kod” [Online]. Available: https://sni2007.scb.se/default.asp. [Accessed Apr. 19 2023].

[12] imbalanced-learn “SMOTE“ [Online]. Available: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html. [Accessed May 04, 2023].

[13] scikit-learn “sklearn.svm.LinearSVC” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC. [Accessed May 04 2023].

[14] J. Brownlee. “Controlled Experiments in Machine Learning” (June 22, 2018). [Online] Available: https://machinelearningmastery.com/controlled-experiments-in-machine-learning/. [Accessed May 08 2023].

[15] P. Langley “Machine Learning as an Experimental Science” Machine Learning 3, p 5-8 (1998). [Online] Available at Springer Link: https://link.springer.com . [Accessed May 08 2023].

[16] V. Jayaswal. “Laplace smoothing in Naïve Bayes algorithm.” (Nov. 22, 2020). [Online]. Available: https://towardsdatascience.com/laplace-smoothing-in-na%C3%AFve-bayes-algorithm-9c237a8bdece [Accessed June 5, 2023].

[17] “Stemming and lemmatization” [Online] Available: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html [Accessed June 5, 2023]

A Appendix 1

The source code for our Jupyter Notebook application can be accessed at either of these two GitHub repositories:

https://github.com/Erkabubben/machine-learning-sni-code-from-company-description

https://github.com/Sobaze/machine-learning-sni-code-from-company-description