MASTER'S THESIS | LUND UNIVERSITY 2016

Product Classification - A Hierarchical Approach

Mikael Karlsson, Anton Karlstedt

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2016-31

Product Classification - A Hierarchical Approach

(Classifying products to compute their socio-ecological impact)

Mikael Karlsson
gda10mka@student.lu.se

Anton Karlstedt
ada08aka@student.lu.se

August 9, 2016

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Pierre Nugues, Pierre.Nugues@cs.lth.se
Kristian Rönn, kristian@normative.io

Examiner: Elin Anna Topp, elin_anna.topp@cs.lth.se

Abstract

The social and environmental impact associated with consuming a product is something that is becoming increasingly important to consumers and businesses alike. This impact can in theory be computed simply by classifying a product and by mapping the classified product to the corresponding life-cycle assessments research. However, this type of mapping requires an extensive product taxonomy with a large plurality of categories, which makes classification through machine learning a non-trivial task.

This thesis describes the implementation of a hierarchical product classifier using the Python library scikit-learn, that makes it possible to automatically classify products based primarily on their brands and titles into a large taxonomy.

Using a data set of 3.1 million products spread over 800 categories, we trained a set of hierarchical classifiers using different learning algorithms. Evaluations of these algorithms showed that the best hierarchical classifier reached a hierarchical F_1-score (hF_1) of 0.85.

Keywords: product classification, hierarchical classification, text classification, UNSPSC, GTIN, scikit-learn

Acknowledgements

First of all we would like to thank our supervisor Pierre Nugues for lending us his wisdom and giving us much positive feedback during this thesis project. We would also like to thank Kristian Rönn, Robin Undall-Behrend, Adam Wamai Egesa and Christoffer Olsson at Meta Mind, without which this thesis would not have been possible.

Finally, we especially thank Kristian Rönn for his brilliance, insightful comments and excel wizardry and Adam Wamai Egesa for many fruitful machine learning discussions.

Contents

Chapter 1
Introduction

This chapter describes the background that led to this thesis, as well as the problem that was solved.

1.1 Background

Each time a product is consumed, it has a socio-ecological impact on our world. The socio-ecological impact that the consumption of a product entails is complex and depends on many factors, such as what materials were used in production, how they were harvested, and the working conditions of the people employed in production, to name a few. This complexity means that to be able to compute the socio-ecological impact, many factors with different measures have to be taken into account, for instance CO_2 emissions, land use, and health outcomes (measured in quality adjusted life-years (QALY)). Furthermore, each product consists of different materials and is produced in different ways, which makes it even more difficult to compute this impact.

Computing the overall impact of products and services in a systematic way through sustainability assessments is a vibrant field within environmental engineering. Despite this fact, the respective results are poorly disseminated to the general public and are thus rarely used in consumer decision making. One way to simplify and generalise the dissemination of this research would be to automatically classify a product, record the quantity consumed and map it to the corresponding sustainability assessments.

To compute the impact of consuming a product within a specific category using this simplification, we need three things:

Category: The category of the product being consumed, for example *food* or *clothing*, or preferably more specific categories such as *butter* or *jeans*.

Impact factors: A list of factors, derived from sustainability assessments, describing the impact per unit of a product from the specified category, for instance CO_2 per kg, or

QALY per USD.

Amount: The amount that was consumed of the product, such as 1 kg or 10 USD.

This however, needs a large set of possible categories, especially if the goal is to be able to perform these computations for all possible products in the entire world. This large number of categories brings a new problem to light, which is:

> How can we automatically decide what category a product belongs to, given a large set of possible options?

It is this question this thesis tries to answer.

1.2 Problem description

The purpose of this thesis was to investigate how products can be automatically classified into a large taxonomy in order to compute their socio-ecological impact, by utilising a pre-existing database specifying the impact factors of the taxonomic categories, and to implement a prototype classifier using machine learning techniques.

1.3 Contributions

By combining the result of previous scientific research as well as a pre-existing database for computing the socio-ecological impact of product categories, this thesis presents a way of computing the impact of products based on products' titles and barcodes.

More specifically, this thesis details how a hierarchical classifier primarily based on product brands and titles can be used to classify products into the United Nations Standard Products and Services Code (UNSPSC) taxonomy. The hierarchical classifier reports a high success rate as compared to previous methods. We also train and compare hierarchical classifiers using several different learning methods.

The work has been divided evenly between the authors to the best of our ability, although some parts of the work have naturally been made by the individual that felt more comfortable with the task at hand and could thus work faster.

Mikael has focused more on writing the report, while Anton has been focusing more on collecting data, investigating, and combating the imbalance of the data set. Anton took more part in setting up and evaluating the UNSPSC Segments experiments as well as implementing the majority of the feature extraction pipeline, while Mikael put more focus on implementing and evaluating the hierarchical classifier.

1.4 Previous work

Previous work in the field can be divided into two main parts: The first one dedicated to classifying products, and the second to compute the socio-ecological impact.

1.4.1 Product classification

Product classification is to automatically divide products into a set of categories or *classes*, something that has become increasingly important as businesses have begun to share more data between each other electronically (Ding et al., 2002; Abels and Hahn, 2006). Ding et al. (2002) trained a classifier to predict products into an older version of the UNSPSC-taxonomy. Their classifier consisted of four naive Bayes models, one for each of the UNSPSC levels. Using this approach, they achieved an accuracy of 78%, although their model was based on vector space models of full product descriptions, something we in this thesis do not have.

In Abels and Hahn (2006), the authors provided a semi-automatic solution to the *reclassification* problem, i.e., reclassifying an already categorised product into another taxonomy, by using machine learning techniques and especially exploiting the existing category of a product. Furthermore, they employed a multitude of classifiers, and filters to try to figure out which category is best suited for a product. They managed to achieve a 76% accuracy in their classification attempts, which was coupled with a manual step where a human decided the final category of the product.

Wolin (2002) classifies products in smaller slices of the UNSPSC-taxonomy by constructing vectors based on words that occur in each category of the training set, and computed the cosine similarity measure between an input product feature vector and all candidate categories. They focused especially on a subset of 272 categories, and reported a test accuracy of up to 79.5%.

Kim et al. (2006) focused on similarities between term frequencies of specific product attributes and their values instead of texts which describe products, and managed to train a naive Bayes classifier with an accuracy of up to 81%.

In more recent years, Yu et al. (2013) described a tool for short-text classification based on a study (Yu et al., 2012) of how to optimally pre-process and extract features from short texts. The tool was able to classify 10 million products based on only their titles into 34 categories with over 90% accuracy.

1.4.2 Other

Kiritchenko et al. (2006) implemented a hierarchical classifier to perform text classification and most relevant to this thesis developed an evaluation measure that supposedly is more suitable in the hierarchical classification context in order to measure model performance. They managed to train an hierarchical text classifier which could predict documents into 1,000 categories using Adaboost with a hierarchical F_1 score of 59.27%. It is this measure we used in our thesis to evaluate the trained classifiers.

Recently, Wamai Egesa (2016) presented a way of computing the socio-ecological impact by classifying transactions and by utilising the same system for computing the actual impact. In his thesis, the trained classifier predicts transactions' Merchant Category Codes (MCC), which are then mapped to distributions of United Nations Standard Products and Services Codes (UNSPSC), that are then used to compute the final impact of the transaction.

1.5 Report structure

From this point onward, the report is structured into five major chapters: Method, Data collection, Implementation, Evaluation, and Conclusions:

- *Chapter 2. Method* introduces important topics related to the problem and the proposed solution.
- *Chapter 3. Data collection* describes the data gathered to perform the experiments.
- *Chapter 4. Implementation* provides a description of the hierarchical product classifier that was implemented.
- *Chapter 5. Evaluation* describes the experiments used to measure the classifier's performance, as well as the result from these and a discussion of them.
- *Chapter 6. Conclusion* tries to summarise the result, and proposes future work and possible improvements.

Chapter 2
Method

As previously stated, the problem we attempt to solve in this report is classification of products into a large taxonomy. We begin this chapter with an introduction of the taxonomy used, UNSPSC, and give an introduction of the Global Trade Item Number (GTIN), an identifier for products which hopefully contains predictive information of their category, and conclude with information about text classification and the system used for computing the socio-ecological impacts.

2.1 The UNSPSC taxonomy

The United Nations Standard Products and Services Code (UNSPSC) is a taxonomy that was initially created by merging the United Nations Common Coding System (UNCCS), and the Dun & Bradstreet's Standard Product and Service Codes (SPSC) in 1998 (Grenada Research, 2001). It provides a logical framework for classifying goods and services, and is currently governed by the organisation GS1 (GS1, 2016). Updated versions of the taxonomy are released regularly and version 17, which is the version we target in this thesis, contains over 65,000 categories.

The UNSPSC taxonomy is constructed as a tree structure with four levels called *Segment*, *Family*, *Class* and *Commodity* (Grenada Research, 2001). Each *node* in the tree is assigned a textual description as well as a two digit combination which is unique with respect to its parent. Following a branch from the top of the tree, the contextual category of each step is narrowed, making it easy to find a product in the taxonomy. Furthermore, the segments can be divided into five groups which are ordered in a way that represents how value is added to products in the supply chain. The groups can be seen in Table 2.1 and in this thesis, we are primarily interested in the products that belong within segments 42-60, as most consumer products fall within these segments.

To reference a certain node in the tree, the two digit combinations of the parent nodes are concatenated to form a unique eight digit combination. If the concatenated string is

Table 2.1: UNSPSC segment groups

Range	Group name	Number of categories
10-15	Raw materials	9614
20-27	Industrial equipment	3712
30-41	Components and supplies	6011
42-60	End use products	39098
70-94	Services	3979

Table 2.2: Example of the UNSPSC Taxonomy

Level	Code	Description
Segment	27 00 00 00	Tools and General Machinery
Family	27 11 00 00	Hand tools
Class	27 11 16 00	Forming tools
Commodity	27 11 16 17	Rubber mallet

less than eight characters, as a result of referencing a node that is not on the bottom of the tree, the string is filled up from the right with zeroes until it is eight characters long.

An example of hierarchical categories in the UNSPSC-taxonomy is given in Table 2.2

2.2 Global Trade Item Number

The Global Trade Item Number is a number intended to serve as a unique global identifier to any trade item, and is usually accompanied by a barcode, which is an encoding of the number that is easily readable by a barcode scanner. The GTIN comes in several different types, which are all referred to by their length. An example of this is GTIN-8 where 8 refers to a length of eight digits (GS1, 2016). The two most prevalent types of GTIN in the retail market space are GTIN-12 and GTIN-13. GTIN-12, also known as UPC is a 12-digit code primarily used to identify products and trade goods in North America.

After the UPC was introduced in the United States, the EAN-code (GTIN-13) was developed by a number of European organisations for use in the rest of the world (GS1, 2016). It is made up of 13 digits and was designed as a superset of UPC, allowing the two to be used side by side (GS1, 2016). The EAN-code was later given the name GTIN-13. GTIN-8 is designed to be used on trade items that are too small to fit the barcodes of the larger types (GS1, 2016), and is primarily used outside of North America. GTIN-14 is an extension of GTIN-13, which adds information regarding packaging size for trade items not intended to be sold at retail (GS1, 2016).

A GTIN is generally defined by three parts; a company prefix, an item reference number and a check digit, with the exception of GTIN-14 which also has a indicator digit preceding them (GS1, 2016). The overall structure of the GTIN-12 and GTIN-13 can be found in Table 2.3.

Table 2.3: GTIN General Structure

Type	GS1 Company prefix grows ⟶	Item reference ⟵ grows	Check digit
GTIN-8	0 0 0 0 0	$N_1N_2N_3N_4N_5N_6N_7$	N_8
GTIN-12	0 $N_1N_2N_3N_4$	$N_5N_6N_7N_8N_9N_{10}N_{11}$	N_{12}
GTIN-13	$N_1N_2N_3N_4N_5$	$N_6N_7N_8N_9N_{10}N_{11}N_{12}$	N_{13}

Table 2.4: Select GS1 Prefixes

GS1 Prefix	Significance
00001 - 019	Extended UPC Company Prefixes
20 - 29	Used to issue GS1 RCN's within a geographic region
300 - 976	Used to issue GS1 Company Prefixes
977	ISSN (Serial Publications)
978-979	ISBN (Books)

2.2.1 Company Prefix

The company prefix is a four to twelve digit number assigned to a company by one of the GS1 member organisations in order for the company to generate GTIN's for their trade items. The first digits of the company prefix must be one of the GS1 prefixes.

The GS1 prefixes have different meanings and are used for different purposes. Some are reserved for generating region or company specific GTIN's, called Restricted Circulation Numbers (RCN), which are not unique global identifiers. The majority of the prefixes however have been distributed to GS1 member organisations for use in distributing company prefixes. This implies that a company prefix must be at least one digit longer than its GS1 prefix. Table 2.4 shows what the various groups of GS1 prefixes are used for.

As the length of the company prefix can vary while the length of a given type of GTIN is fixed, the company prefix effectively determines how many trade items a company may generate GTIN's for. A short company prefix for example, lets a company use more digits to specify the item reference, which yields a higher number of item reference combinations. The variable length of the company prefix also means that given a GTIN, it is not possible to programmatically determine what part of the GTIN is the company prefix and what is the item reference. Fortunately, there are publicly available data describing the relationship between the length and the initial digits of company prefixes (GS1, 2016).

For example the GTIN 7310400020731 is a Swedish Loka water bottle. It has a GS1 prefix of 73, which refers to the Swedish GS1 member organisation. Moreover, it has the company prefix 731040, referring to a specific company. The length of that company prefix limits the amount of products that the company can distribute using that company prefix to 10^6 different products.

2.3 Machine Learning

Machine learning tasks can generally be divided into two categories – supervised learning, and unsupervised learning. In supervised learning, a *model* is trained to describe a known

Table 2.5: Unigram vector space model of the text in the product names (1) *Hello Kitty Computer Mouse*, (2) *Hello Kitty Lunch Box* and (3) *Wireless Mouse*

index	{Hello}	{Kitty}	{Computer}	{Mouse}	{Lunch}	{Box}	{Wireless}
(1)	1	1	1	1	0	0	0
(2)	1	1	0	0	1	1	0
(3)	0	0	0	1	0	0	1

relation between a set of input and output variables. The set of samples from which a model is taught is called a *training set*, and each sample in a training set D consists of a set of input variables $\mathbf{x} = \{x_1, .., x_n\}$, and an output variable y.

The goal of supervised learning is to fit a function f to training set D so that it describes the relation $\mathbf{x} \cdot f = y$ as accurately as possible for the entire training set. This function f can after training be used on samples where the output variable is unknown, to *predict* the outcome of a certain event. Furthermore, if the range of f is discrete, the task is called a *classification* task, and if the range is continuous, it is called a *regression* task.

Text classification, or text categorisation, is the task of extracting *features* from a text in order to decide what the topic of the text is, given a number of possible topics. The topics are formally called *classes* and the scope of a class may be broad or narrow, depending of the task at hand. A typical text classification task is to classify news articles, and the classes may then be broad topics such as politics or sports, or more narrow topics such as specific current world events.

2.3.1 Text features

Text classification decisions are based on *text features*, which are any items found in a piece of text that can be used to distinguish one piece of text from another, and a typical example of text features are *N-grams*. N-grams are sets of length N made up from sequences of words or characters that occur in pieces of text. As an example, the unigrams (or 1-grams) of the text in the product name *Hello Kitty Computer Scroll Mouse* are: $\{Hello\}$, $\{Kitty\}$, $\{Computer\}$, $\{Scroll\}$, $\{Mouse\}$. After extracting these sets from pieces of text, the frequency of their occurrence can be computed and represented as a *vector space model*, as illustrated in Table 2.5. These vectors can then be used to compute the cosine similarity measure, which can be seen in Equation 2.1, where $\mathbf{x}_1$ and $\mathbf{x}_2$ are two input vectors.

$$k(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1 \mathbf{x}_2^T}{||\mathbf{x}_1|| ||\mathbf{x}_2||} \tag{2.1}$$

The cosine similarity measure describes how similar two vectors are to each other in one number, and therefore how similar two pieces of text are.

2.3.2 Feature hashing

The basic approach to converting text tokens such as N-grams into a vector space model is to map each token in the training set to a column index. This can lead to problems when the training set is large, as the map between tokens and columns has to be kept in memory.

A more efficient way of vectorising the tokens is to use a hash function to calculate the indices of the tokens. This is done by creating a vector of n columns and a hashing function that yields n possible values, and applying this hashing function to each of the tokens in the string. This is called feature hashing, or "the hashing trick." The main downside of this approach is that if an insufficient size n is chosen or a bad hashing algorithm is used, collisions can occur between tokens, which means that the hash codes of two different tokens result in the same column value. This makes it impossible to distinguish between the two tokens.

2.3.3 Stemming

Stemming is the practice of reducing words to their word stem, words that for grammatical reasons have been extended (Manning et al., 2008), something which is done a lot in English. This extension can sometimes disrupt similarity comparisons between words. For instance, the words *inflated* and *inflate* have essentially the same meaning, but when represented as unigrams, they would be two separate words without any relation.

Stemming can sometimes restore this lost relationship, but it does not always yield positive results, as noted by Yu et al. (2012), who argue that stemming is especially harmful in classification of short text, where the form of words are more meaningful. Furthermore, Russel and Norvig (2014) state that stemming usually results in a 2% increase in recall rate, but that it can harm precision. Furthermore, stemming is only suitable for processing English texts, and more advanced techniques exist today such as lemmatisation, where words are transformed using a dictionary.

2.3.4 Learning methods

In order to find the function f from a training set, there are a number of different *learning methods* to employ. Learning methods take the training set D as input and output the function f, which in the case of this thesis would be a function that considers the features of products' title, brand and GTIN, and returns a class.

Different learning methods use widely different strategies for finding the function f, all with different strengths and weaknesses. Naive Bayes is one such method, and one of its strengths is its simplicity, so it serves as a good example of a learning method and is therefore outlined below. A description of another learning algorithm called Support Vector Machines, which tackles the problem differently, can also be found below. For a textbook example of either algorithm, see Russel and Norvig (2014).

Naive Bayes

Naive Bayes classifiers are based on Bayes theorem which states the following relation between a class y and a feature vector $\mathbf{x} = (x_1, ..., x_n)$:

$$P(y|\mathbf{x}) = \frac{P(y)P(\mathbf{x}|y)}{P(\mathbf{x})} \tag{2.2}$$

where $P(y|\mathbf{x})$ is the probability that input vector $\mathbf{x}$ belongs to output class y, $P(y)$ the prior probability of y, $P(\mathbf{x}|y)$ the occurrence of $\mathbf{x}$ in class y, and $P(\mathbf{x})$ the overall occurrence of

input vector **x**. The theorem is then simplified by making the "naive" assumption that the features are conditionally independent

$$P(y|\mathbf{x}) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(\mathbf{x})} \tag{2.3}$$

and since the denominator is constant in relation to the input vector, the expression can finally be simplified into:

$$P(y|\mathbf{x}) \propto P(y)\prod_{i=1}^{n} P(x_i|y) \tag{2.4}$$

where $P(x_i|y)$ is the probability of feature x_i occurring in samples **x** belonging to output class y, and $P(y)$ the proportion of training samples belonging to the output class y.

Support Vector Machine

Support Vector Machines (SVM) take a different approach than Naive Bayes. Instead of finding the function f by computing probabilities of the features that make up the input vector space models, SVM's treat each input vector of length N as a point in an N-dimensional space.

In the binary classification case, the points of the two classes can be separated linearly from each other by a decision plane. To compute the position of this plane, a number of points from each class are used. These points are called *support vectors*, and the decision plane can be computed from these into the function f for predicting the class of an unknown input vector. This makes SVM's efficient as they only have to consider a subset of the input vectors to find a solution.

In the case where the two classes cannot be cleanly separated by a plane because points of the two classes overlap, a soft margin can be used by allowing points to overlap, but punishing those points based on their distance from the decision plane according to a specific function.

Another approach to when classes cannot be linearly separated is to map the support vectors to some other Euclidean space of higher dimension using what is called a kernel function. By performing this mapping, a linear plane separating the two classes can always be found, which in the lower dimensional space results in a non-flat surface separating the points.

For multi-class problems, multiple binary "One-versus-rest" SVM's are usually trained from the data, which is why we only describe a binary Support Vector Machine here. For more detail on the theory behind SVM's and how they may be implemented, see Burges (1998).

Other learning methods

Finally in detailing the learning methods, there are three other groups of methods that are evaluated; *Decision Tree*, *Random Forest* and *Logistic Regression*. They will be limited to a brief description since this section mainly intends to serve as an introduction to machine learning methods. Textbook examples of these can also be found in Russel and Norvig (2014).

- *Decision Tree* methods build models that consist of a set of questions, ordered in a tree. A decision tree is learnt by splitting a training set into smaller subsets based on features and their values, and at each split, a question is generated. This process is repeated until a stage is reached where all the samples in a subset are of the same class. These questions can later be used to predict the output class a sample should be assigned to.

- *Random Forest* methods are based on Decision Trees, but introduces some randomness to the classification. The general principle consists of randomly dividing the training set into smaller subsets where each set is used to train a small separate decision tree. The output class of a sample is finally decided by letting a number of randomly selected decision trees classify the sample. The average answer of these classifications is then selected as the output class.

- *Logistic Regression* is in its essence linear regression, where a linear function that describes the relationship between the input variables (features) and the output variable is derived from a training set. However, instead of predicting a continuous value it predicts the probability of a certain discrete event by using a logistic function. The event that is predicted in the classification context is the belonging of a specific sample to a certain output class.

These algorithms are some of which supported by the machine learning library Scikit-learn that was used in this thesis. They were chosen out of curiosity of how well different types of learning algorithms are suited within the hierarchical classification setting and how their results differ, which we try to evaluate in this thesis.

2.3.5 Evaluating classifier performance

When evaluating classifiers, a common practice is to divide the data into two parts, one for training the classifier, the aforementioned *training set* and a second part called *test set* that it used for testing it. This testing is done by using the classifier to predict the classes for the samples in the test set and comparing them to their actual classes.

With this comparison there are several measurements that can be used, this thesis will use and focus on three; Precision, Recall and F_β score. They are defined below as measurements for a specific class, but this can later be expanded to cover the entire test set by averaging the measurements from all classes.

- *Precision* for a given class is the ratio between the amount of the instances that were correctly predicted to it and the total amount of instances that were predicted to it.

- *Recall* for a given class is the ratio between the amount of the instances that were correctly predicted to it and the total amount of instances that should belong to that class.

- F_β *score* is a mean of two above, where β refers to a constant that weighs the importance of one of the measurements. If β set two one, that means that precision and recall are equally important and is referred to as F_1. It is defined as:

$$F_1 = 2\frac{precision * recall}{precision + recall} \tag{2.5}$$

2.4 Socio-ecological impact database

To compute the socio-ecological impact, a database developed by Meta Mind AB was utilised. It contains scientific research and sustainability assessments detailing the impact factors of various product categories in the UNSPSC-taxonomy, which is why UNSPSC is the target vector for the classification in this thesis. The impact factors come from a number of different sources and are computed in a number of different ways, some of which are outlined in Table 2.6, and in this database impact factors are defined as a relationship between a consumption unit of a category and an output impact. For example, an impact factor of jeans could be defined the following way: 0.02 kg SO_2 per USD jeans consumed.

The surrounding system also contains functionality for performing calculations based on a measure of consumption of a product in a certain UNSPSC which are available through an API.

Table 2.6: The research data that make up the impact factors

Type of research	Description
MRIO	**Multi-regional input-output** analysis (MRIO) is a method for calculating the social or environmental impact of various economic sectors within a given country. This is done by following how money and resources flow between different sectors in the economy(Miller and Blair, 2009). For example, to find out the greenhouse gas emissions of the Swedish real estate sector, one could look at how much money that entire sector spent on buying energy from the energy sector.
LCI	A **life cycle inventory** database (LCI) is a type of database that contains information on the environmental impact of a specific production process that are a part of a product's lifecycle(McDonough and Braungart, 2002). For example, an LCI could contain information on the environmental impact from harvesting grains.
LCA	A **life cycle assessment** (LCA) is a method of calculating the social or environmental impact of products or services. A complete LCA is essentially performed by adding all LCI factors for the product's entire life cycle from cradle to grave (e.g., from material extraction through transportation, production and and waste disposal)(McDonough and Braungart, 2002).
EPD	An **environmental product declaration** (EPD) is simply a specific type of LCA done by a company with the purpose of analysing the impact of one of their products. An EPD can be done for many reasons, such as public relations, following laws and regulations or for improving production processes.

Chapter 3
Data collection

As explained earlier, classification through supervised learning requires a set of labelled training samples to train a classification function. Generally a larger training set is the best way to increase the future performance of the classification. In this chapter, we describe how we collected the data set.

3.1 Open data sets

There are a number of open data sets online, with varying content quality which can be used for training a classifier. The open data sets used as part of this thesis are described here.

3.1.1 Icecat

Icecat is a global syndicate that produces product data sheets in several languages, and provides product statistics for use on its clients' e-commerce sites such as shopping sites, product reviewing and comparison sites, or for other purposes (Icecat, 2016). Most of the data sheets by Icecat concern electronic products, but there are also sheets for other products, such as office supplies, beauty and toy products. Paying customers have access to the full database which is updated daily and contains approximately three million data sheets and nearly 1,300 brands, while non-paying customers have access to a smaller, free part of the data set. Each data sheet describes one product and its variations, which means that each one data sheet may contain more than one GTIN. For example *Toshiba Battery Pack (Li-Ion 9 cell 4,500mAh)* has two GTINs associated with it, 842740007297 and 410000203538.

The data obtained from Icecat contained a total of 9,900,000 unique products including their variations or models, of which about 5,100,000 did not have any category specified. The rest of the products which amounted to about 4,800,000, were categorised into almost

2,800 textual categories, for example *notebooks* or *keyboards*. Only the products that had a category assigned were of interest to us as we wanted to use this information as training data for our models and in our experiments.

3.1.2 Open Food Facts

Open Food Facts is a collaborative project with the goal of creating a comprehensive open database of food products and their nutritional information, which anyone can contribute to. As of writing this thesis, the database contains approximately 100,000 products of varying detail.

Just like the data from Icecat, the Open Food Facts-data were categorised into textual categories. However, as the database is collaborative and does not come from suppliers like the Icecat-data, consequently the categories used were less standardized. Almost 75,000 products, divided into almost 2,200 categories, could be extracted from the data set.

3.1.3 Open Product Data

Open Product Data is a public database of over 900,000 products and their associated brands, which can be downloaded in its entirety. The Open Product Data-project is governed by the Open Knowledge Foundation, which is a non-profit network of individuals with the vision of making as much knowledge and information available to as many people as possible.

The Open Product Data-database contains the data from Open Food Facts, but this data was to a large extent missing and erroneous, which is why it was downloaded separately. Furthermore, of the 900,000 products in the Open Product Data-data, approximately 400,000 contained a product name, and were categorised into 31 top-levels of the GPC-standard. The GPC standard is much like UNSPSC a four level taxonomy, however, the two do not overlap and complement each other, which makes it difficult to convert data from one to the other.

3.2 Labelling data

As the data obtained from the three data sources were not categorised according to the UNSPSC-taxonomy, they could not be used for training models directly. Instead, the data had to be labelled appropriately. This section details how this was done.

3.2.1 Cleaning textual categories

The data obtained from Icecat and Open Food facts, both of which had textual categories, were refined using the tool Open refine that was previously developed under the name Google Refine, until 2013 when involvement with Google stopped (Huynh, 2013). The purpose of this refinement was to merge duplicates and similar categories (for example, “Bottled soda” and “soda” reference the same thing despite being written differently).

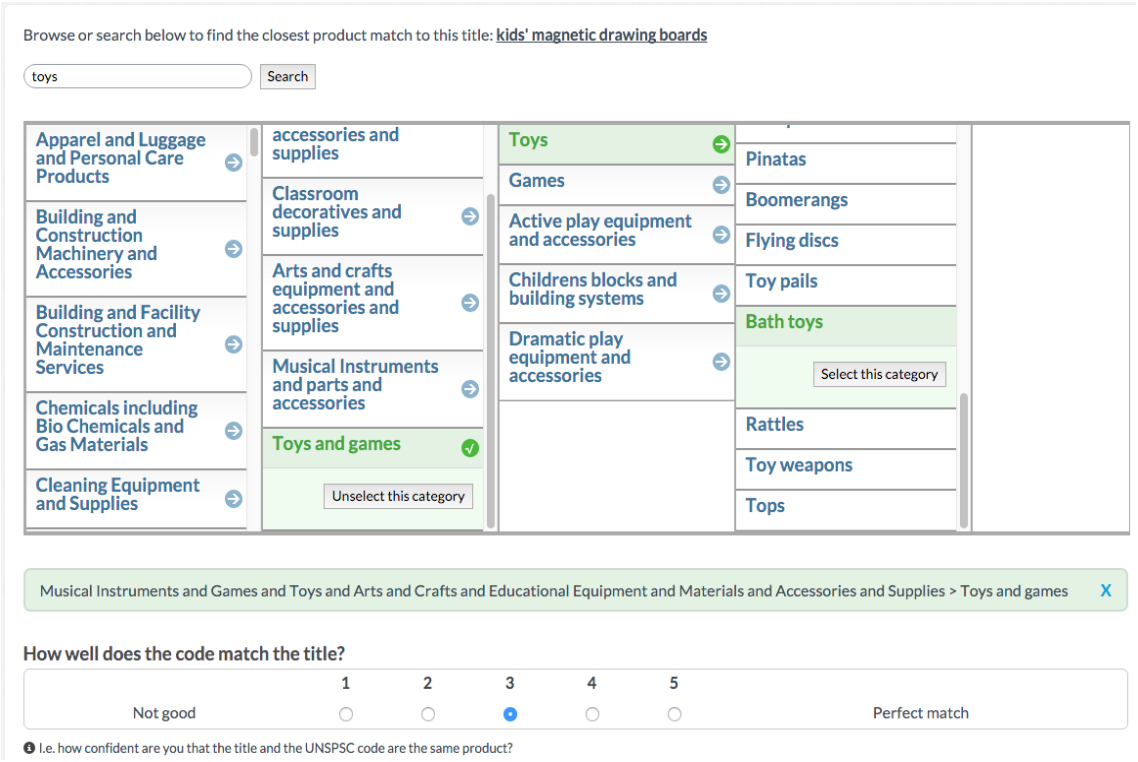

Figure 3.1: CrowdFlower job overview

To begin with, a histogram of how products were divided into categories was created, after which different built-in tools were used to try to cluster similar categories. The built-in clustering tools used were Levenshtein distance, a method for calculating differences between strings, k-Nearest Neighbours based on N-grams, as well as Metaphone 3, which is an algorithm that tries to approximate a phonetic key for English words that sound similar. After clustering categories and dropping categories with less than 10 examples, 2,413 unique text categories remained from the original. This refining of categories decreased both time and cost of converting the categories into the UNSPSC-taxonomy, described in the following step.

3.2.2 Amazon mechanical turk

CrowdFlower is a platform we used for easily crowd-sourcing simple tasks through Amazon's Mechanical Turk programme, where individuals are paid for performing small, almost *mechanical* tasks. This was used in order to convert the textual categories into their UNSPSC-equivalents. The CrowdFlower job was set up so that it allowed five individuals to browse the UNSPSC-taxonomy and select the best matching UNSPSC for each of the categories that had been extracted.

To validate that the results were of high quality, a quiz was designed where individuals were randomly asked to classify one of 60 products that had already been classified by us. If an individual failed to classify 80% of the products in the quiz, the individual was not allowed to continue, and their previous classifications were discarded. Figure 3.1 illustrates what the job looked like to the participating individuals.

When the job had been completed, the answers were aggregated by selecting the UNSPSC that most individuals had chosen weighted by their *meta-confidence*, a measurement of their quiz-score and their overall score on other CrowdFlower jobs. The answers were then added to the data sets by joining the categories with the answers, to create labelled data sets.

3.2.3 Manual labelling

As the Open Product Data-dataset did not contain any textual categories, the same methodology as for the other data sets could not be applied to it. It was therefore decided to manually label some of these data. However, this was deemed far too resource-demanding, but in the end, approximately 90,000 products were classified into 15 top level categories (i.e. the "segment" level of UNSPSC).

Chapter 4

Implementation

This chapter details the implementation of a greedy hierarchical product classifier. We used python as programming language as well as the scikit-learn machine-learning library. We begin by giving a brief description of the tools and libraries used, and the overall concept of the hierarchical classifier. We then explain the flow of events during training and predicting. Lastly, we describe the feature extraction pipeline and the evaluation measure implemented to evaluate our classifier.

4.1 Tools and libraries

The hierarchical classifier and the training tool were implemented with the help of a number of libraries, which are detailed in this section.

4.1.1 Pandas

Pandas is a data analysis toolkit based on Python which makes it possible to access and edit tabular data in a fashion similar to that of a relational database (McKinney and PyData Development Team, 2016). Pandas' two basic data structures are Series and Dataframes, which are one- and two-dimensional respectively. The latter can essentially be seen as a table in a relational database, as it supports much of the same functionality such as selecting and inserting, or more complex tasks such as grouping and joining data sets based on column values.

Pandas is used extensively during training and in usage of the hierarchical classifier in order to clean, modify, merge, and select data based on different criteria, such as variations of existing UNSPSC's in the data sets.

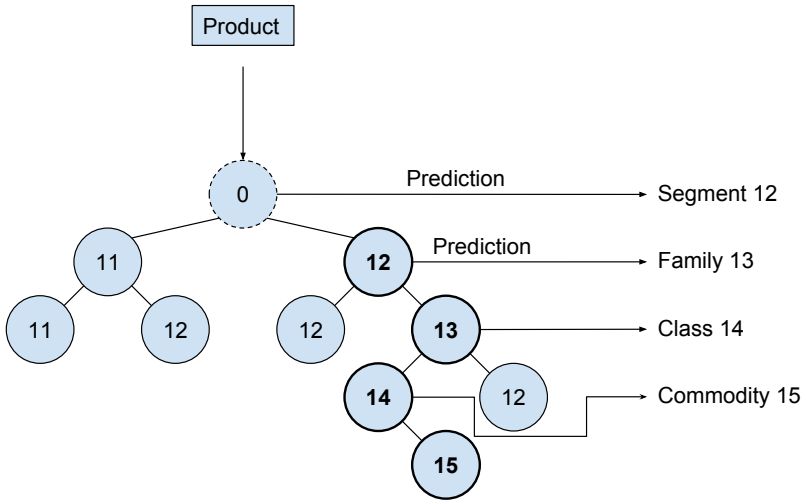

Figure 4.1: Prediction of product into category 12131415

4.1.2 Scikit-learn

Scikit-learn is an open source machine learning library written in python, with many algorithms for supervised and unsupervised learning and a variety of feature extraction and feature selection methods (Pedregosa et al., 2011). Scikit-learn incorporates the same pre-compiled libraries that LibShortText (Yu et al., 2013) is built upon, LibSVM and LibLinear, and uses the package Scipy for performing numerical computations. Scipy has bindings to low-level numerical computations implemented in Fortran (Pedregosa et al., 2011), which allows scikit-learn to be both fast and easy to use at the same time.

4.2 The hierarchical classifier

The hierarchical classifier developed as part of this thesis project makes it possible to perform classification of products into large taxonomies that would otherwise be difficult because of the large number of classes that have to be considered.

The key concept of the hierarchical classifier is to exploit the hierarchical structure of the UNSPSC taxonomy to break down the classification problem into several smaller parts, much like how a human would do. A human would not begin by sifting through all 60,000 possible commodities (leaf categories) at the bottom of the taxonomy, but would instead look at the segments (top-level categories) which are only 56. After a suitable segment is found, its child categories would be considered and selected. This process would be repeated until a commodity is reached or there are no suitable child categories left, which means that the final category for the product has been found.

The hierarchical classifier implemented works in the same way by employing multiple classifiers, one at each node in the UNSPSC taxonomy that is not a leaf, and one at a fictitious *root* node in order to predict an initial UNSPSC segment, as can be seen in Figure 4.1

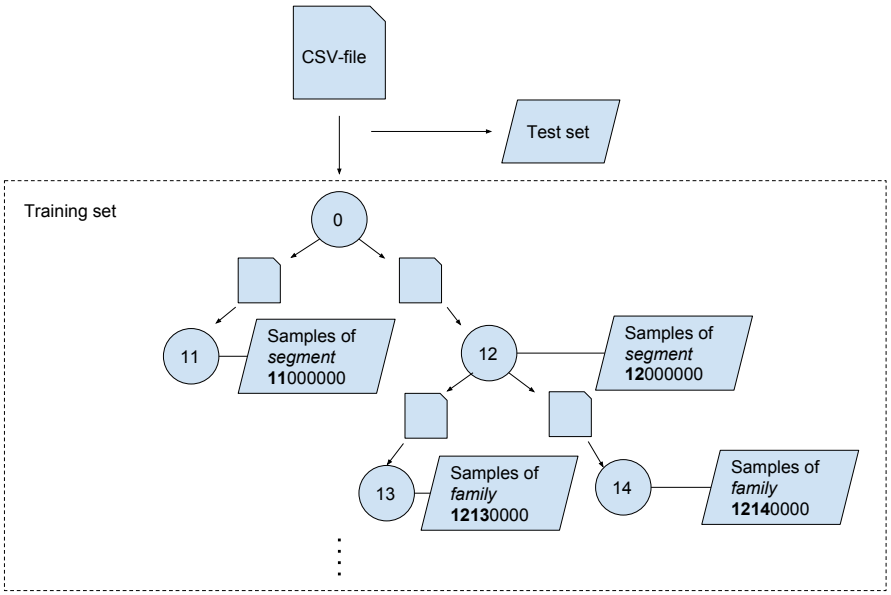

Figure 4.2: The training set is recursively divided into *segments*, *families*, *classes* and *commodities*.

4.2.1 Training a hierarchical classifier

The hierarchical classifier is trained in two steps.

1. In the first step, a file containing the collected samples of product titles, brands and UNSPSC's is read, by using the pandas library, and a DataFrame is created from it. The order of the samples in the DataFrame is randomised and the DataFrame is then split into a training set and a test set. A tree that resembles the UNSPSC taxonomy is then created from the samples of the training set by recursively dividing them into their UNSPSC *segment*, *family*, *class* and *commodity*, which is illustrated in Figure 4.2. Each node in the tree is an instance of the `Node` class which has a UNSPSC and a DataFrame containing its samples as well as a list of child nodes. By structuring the training set in the same way as the UNSPSC taxonomy, it is easy to traverse the tree and train models on subsets of the original training set.

2. The second step is to train the models that make up the hierarchical classifier. This is done by traversing the tree in post-order, passing along an instance of the `HierarchicalClassifier` class which has a feature extraction pipeline as well as an instance of a scikit-learn learning algorithm, such as Naive Bayes or SVM for instance. A model is trained for each node that has child nodes, using the child nodes' DataFrames as training data, and their UNSPSCs as output classes. Each trained model is then stored as key-value pairs in a dictionary of the `HierarchicalClassifier` instance, where the key is the node's UNSPSC, and the value is the trained model. An illustration of this process be seen in Figure 4.3.

When the Hierarchical classifier has been trained, it is tested using the test set from step 1, and its performance is measured using the evaluation measure detailed in Section 4.3.

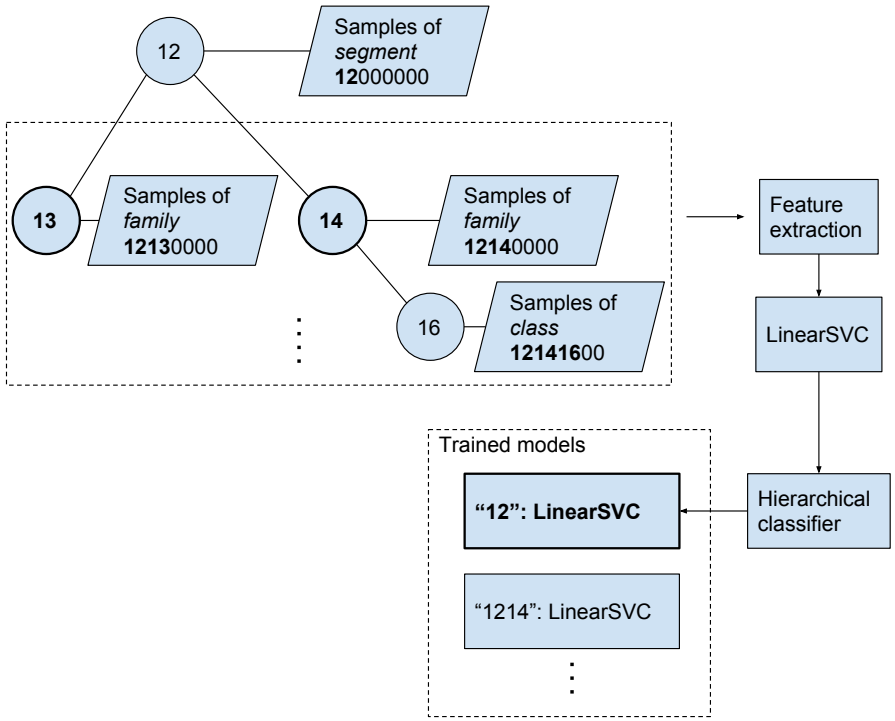

Figure 4.3: A model is trained for segment 12, with output classes 13 and 14, using training data from UNSPSC segments 1213, 1214, and UNSPSC class 121416.

4.2.2 Predicting UNSPSC categories

To predict the UNSPSC category of one or more products, the products are appended to a pandas DataFrame and passed as an argument to the `greedy_predict` function of the `HierarchicalClassifier`. This function will extract features from the products in the DataFrame using the feature extraction pipeline, and will recursively predict the *segment*, *family*, *class* and *commodity* of the products as far down into the UNSPSC-taxonomy as possible. The first model to be used for prediction is always the fictitious root model with key "0", which predicts the segments of the products, although if we for some reason would like to start prediction at some other node in the tree, for instance if parts of the UNSPSC are known, then that prediction could in theory be started at any node.

4.2.3 Feature extraction pipeline

The hierarchical classifier uses a feature extraction pipeline to extract features from a training set. The pipeline takes the training set and converts the samples it contains into vector space models. Each step in the pipeline takes a subset of the training set, and returns a sparse matrix of features, which are finally concatenated into one sparse matrix, after which the matrix vectors are normalised according to their length, as this decreases the training time in some cases (Yu et al., 2012).

The first step of the pipeline is based on the results found by Yu et al. (2012), who found that for short text classification, the optimal preprocessing of input data was to not

perform stemming and stop-word removal, but instead to extract unigram and bigram tokens from the text. They also found that because the words in texts of around 10 words usually are unique, there was no significant difference in prediction accuracy between binary representations of tokens and token frequencies, but that the training time when using token frequencies were slightly longer.

Based on these results, the first step of the feature extraction pipeline of our hierarchical classifier applies a sci-kit learn `HashingVectorizer` to a Brand_Title field of products, which contain the brand and title of the product concatenated. The `HashingVectorizer` extracts unigrams and bigrams from the text and uses feature hashing (explained in Section 2.3.2) to map the binary counts of the unigrams and bigrams to a vector of size 2^{20}.

The remaining steps of the pipeline are specifically designed to exploit the properties of product titles and other attributes such as the information embedded in the GTIN, and are detailed below.

Custom features

There are nine feature deriving steps, all but one of them using simple regular expressions to see if the samples fulfill a condition and being loosely based on word features described by Bikel et al. (1999). The odd one out uses a lookup table to extract the company prefixes from the GTIN of a product.

HasInt extracts occurrences of integers in the title or brand of samples, such as "Bag Cover for *13* Inch Laptop", but not "4.5 inch screen protector".

HasFraction extracts decimal fractions from sample titles or brands like "*4.5* inch screen protector" but not "Bag Cover for 13 Inch Laptop".

AmountOfDigits counts the digit sequences of lengths between 1 and 10 in the title or brand of samples, which make up separate features.

XNumbered marks samples having a number followed by x in the title, e.g., "*2x* packs of cigarettes" or "*2 x* packs of Marlboro cigarettes".

DashNumbered marks any example having a number followed by dash in the title, e.g., "*6-pack* Coca Cola".

AlphanumericWords marks any example having a word containing both number and letters e.g., "Xerox *700i*".

UnitFeatures checks if there are any numbers followed by common units of measurements, e.g., "*2kg* salmon", "Corsair Vengeance *8GB* Desktop Memory". The existence of the different units makes up separate features.

FirstGTINNumber pads the GTIN with zeroes from the left until it has a length of 13 and then returns the first set of digits, describing which country the product was produced in.

CompanyPrefix pads the GTIN of the samples with zeroes from the left until it has a length of 13 and loads a table containing GTIN prefixes and the length of the company prefix of a GTIN with a certain prefix in the table. The prefixes in the table are joined with the GTINs of the samples, to get the length of the individual company prefixes, which are then extracted. Extracted company prefixes are vectorised using feature hashing and returned.

4.3 Evaluation measure

In order to evaluate the classifier's performance, the evaluation measure developed by Kiritchenko et al. (2006) was implemented and used.

The problem with typical measures such as *accuracy* are that they are binary in the sense that if the output class is not the expected one, it is wrong. However, within a hierarchical classification setting, products may be classified correctly in partial. It is for instance possible that a product is correctly classified as medical equipment, but is incorrectly classified in more specific terms. It would of course also be possible for the same product to be classified as something else than medical equipment, which would be more erroneous. Certainly the latter case is more severe, and it is therefore desired to punish these types of misclassifications more severely than the first kind. It is also not uncommon that the hierarchical classifier implemented in this thesis tries to be too specific. This happens when the underlying taxonomy is not precise enough so that products have to be placed in a broader category, one that is not a leaf node in the taxonomy. The classifier implemented as part of this thesis cannot detect if this is the case, and will always try to put it in one of the leaf node categories. An extension that could possibly solve this is presented in Section 6.1.3.

The measure developed by Kiritchenko et al. (2006) takes these phenomenons into account and punishes errors made early in the prediction process more than others. The definition of their evaluation measure can be found below.

4.3.1 Definition

The evalution measure is made up of two new measures, *hP* (hierarchical precision) and *hR* (hierarchical recall), which are defined as follows.

Given a training set of products P and a set of possible classes C, for any product $(P_i, C_i), P_i \in P, C_i \subseteq C$ classified into subset $C'_i \subseteq C$, the sets C_i and C'_i are extended with their ancestor labels (which is something that is done inherently in the UNSPSC-taxonomy): $\hat{C}_i = \{\bigcup_{c_k \in C_i} Ancestors(c_k)\}$, $\hat{C}'_i = \{\bigcup_{c_l \in C'_i} Ancestors(c_l)\}$

The micro-averages of *hP* and *hR* are then calculated:

$$hP = \frac{\sum_i |\hat{C}_i \cap \hat{C}'_i|}{\sum_i |\hat{C}'_i|} \qquad (4.1) \qquad\qquad hR = \frac{\sum_i |\hat{C}_i \cap \hat{C}'_i|}{\sum_i |\hat{C}_i|} \qquad (4.2)$$

These measures are then combined into one hF-measure which can be weighted:

$$hF_\beta = \frac{(\beta^2 + 1) \cdot hP \cdot hR}{(\beta^2 \cdot hP + hR)}, \beta \in [0, +\infty] \qquad (4.3)$$

4.3.2 Example

To illustrate how misclassification at different levels has an effect on the measure, two examples will be given.

For a product that belongs to UNSPSC class $\hat{C}_i = \{60, 14, 10, 10\}$ (toy vehicles) but has been classified into class $\hat{C}'_i = \{60, 14, 10, 25\}$ (toy weapons), $|\hat{C}_i \cap \hat{C}'_i| = 3$ and as a result $hP = hR = \frac{3}{4}$ (three out of four categories are correct).

For another product which belongs to UNSPSC class $\hat{C}_i = \{50, 15, 15, 15\}$ (soy milk) that has been classified into class $\hat{C}'_i = \{50, 17, 20, 01\}$ (soy sauce), $|\hat{C}_i \cap \hat{C}'_i| = 1$ and as a result $hP = hR = \frac{1}{4}$ (one out of four categories are correct).

In both of these examples, both hP and hR are equal, and with a $\beta = 1$, they would be weighted equally in hF and it would thus yield the same result. However, if the hierarchical classifier for some reason cannot classify a product all the way down to a leaf node, which can happen if a model has not been trained for this node because of insufficient examples, hP and hR will be different as the denominator will be different.

Chapter 5
Evaluation

This chapter contains evaluations of the classifiers trained in this thesis project. It starts with a description of the experiments that were conducted together with information about the machine that the experiments were conducted on, as well as the data used. It finally ends with results and discussion regarding them.

5.1 Experimental setup

Two batches of experiments were set up, the first evaluating models trained to classify products into the segment layer of UNSPSC only. The models were trained using thirteen different learning algorithms using different feature extraction pipelines. The second batch of experiments evaluated hierarchical classifiers that consider all layers of UNSPSC, the classifiers were trained using four different learning methods based on results from the first batch of experiments. Both of the experiments measured the learning rate of the trained models, time spent training, and time spent predicting the test set.

5.1.1 Data used

The data used in the experiments were the data collected as described in Chapter 3. These data consisted of 3,102,836 samples distributed across 1,628 different UNSPSCs. This data set was randomised and split into two pieces, one training set with 80% of the samples, and a test set made up from 20% of the samples. The training set was split into five individual training sets, each making up 20%, 40%, 60%, 80% and 100% of the original training samples. This was done to allow for analysis of the *learning rates* of the learning algorithms, i.e., the rate at which the models improve as the amount of samples increase. In both of the two batches of experiments that were set up, one model were trained on each of the training sets, and tested with the same test set.

The data set was not especially rich, and contained the following product attributes:

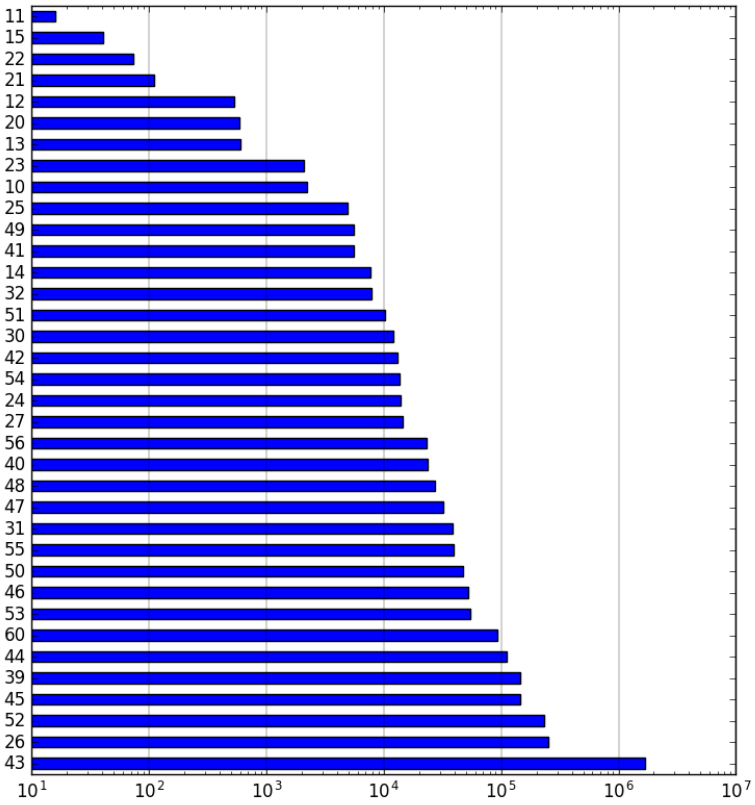

Figure 5.1: Histogram of top level UNSPSCs in the collected data.

GTIN 816,554 (26%) of the samples had a GTIN

Brand All products contained a brand name, for example *Apple*

Title All products contained a title, for example *iPhone 6, 64gb*

unspsc The UNSPSC class of the product.

The brand and title of the products were concatenated into one string to produce a longer field, called Brand_Title.

Figure 5.1 shows the distribution of samples in the data set across their UNSPSC-segments sorted in descending order. It also shows a significant imbalance in the amount of samples available for each class, 59% of the data are part of the largest class: 43 (Information Technology Broadcasting and Telecommunications). This is further explored in Figure 11 in the appendix, which shows a distribution of all the classes and consequently confirms that the imbalance is present on all levels.

5.1.2 UNSPSC segments experiments

The first experiments were performed by extracting the segment, which is specified by the first two digits of an UNSPSC, from the samples in the training set, and by training models using them as output classes. This setup was chosen to give us an initial estimate of how well a model could be expected to perform on product brands and titles and GTIN's alone and if we would have to figure out ways of obtaining more detailed data of products, such as product descriptions.

As this experimental setup would also result in a model that was similar to that of our hierarchical classifier's root model, the scores achieved in this experiment would dictate the maximum score that our hierarchical classifier model could result in. Furthermore, as there were only 31 UNSPSC segments represented in the data set used for these experiments, the results could easily be presented in confusion matrices. A confusion matrix describes the relationship between the predicted classes of the samples in the training set and their actual class, which can be very useful for analysing model performance, especially for imbalanced data sets – data sets are which the data are unevenly distributed over the classes.

The learning algorithms and their scikit-learn classes that were tested in the segments experiments are detailed in Table 5.1. All learning algorithms except the two variants of Naive Bayes (Multinomial and Bernoulli Naive Bayes), which did not support the option, were evaluated using inverse class frequency balancing, by setting a balancing parameter $class\_weight$ = "$balanced$". This uses the training set to weigh classes according to their inverse frequencies, so that classes that occur seldom are favoured more. Models trained using Random Forest and Logistic Regression were trained using option $n\_jobs = 4$, which causes the algorithm to utilise four cores during training.

5.1.3 Hierarchical classifier experiments

In the second experimental setup, the performance of the actual hierarchical classifier was measured. These experiments were conducted much in the same way as the first. A tree was built from 80% of the data set, from which subsets containing 20%, 40%, ..., 100% of the data were extracted and used as training data for the hierarchical classifier, in order to be able to plot the learning rate of it. The hierarchical classifier was then tested using the remaining 20% of the original data set.

It is important to note that the hierarchical classifiers were trained to not consider categories of less than 100 samples, as such classifiers have a tendency to be inconsistent in their scoring, with an average F1-score of less than 0.6 (Lewis et al., 2004). The result of this is that the number of possible output categories dropped from 1,628 to 829.

As opposed to the experiments where only the UNSPSC segments were targeted, no confusion matrices were plotted because of the many output classes, which would create a confusion matrix of 829×829 cells. A confusion matrix of this size would not be particularly useful to plot, as it would be difficult to read. Instead, the primary measurement of these experiments were the hF_1-score of the trained models.

The learning algorithms chosen for these experiments were based upon the results of the UNSPSC segments experiments, and were all trained with inverse class frequency balancing . To investigate if L2-normalisation could shorten the training time of the classifiers, two models were trained on normalised data sets.

Table 5.1: Learning algorithms and their scikit-learn classes used in the segments experiments

Learning Algorithm	scikit-learn class and description
Multinomial naive Bayes	`naive_bayes.MultinomialNB` with smoothing parameter $\alpha = 1$.
Bernoulli naive Bayes	`naive_bayes.BernoulliNB` with smoothing parameter $\alpha = 1$.
SVM	`svm.LinearSVC` with default parameters; penalty parameter $C = 1$, *loss* = "*squared_hinge*", *penalty* = "*l2*", *multi_class* = "*ovr*" (one versus rest).
SVM (CS)	`svm.LinearSVC` with default parameters, but with parameter *multi_class* = "*crammer_singer*", which uses the technique developed by Crammer and Singer (2002) in order to generalise the multi-class problem instead of training several independent binary classifiers.
Decision Tree	`tree.DecisionTreeClassifier` with default parameters; *criterion* = "*gini*", *splitter* = "*best*", *max_depth* = *None*.
Random Forest	`ensemble.RandomForestClassifier` with default parameters; *n_estimators* = 10, *criterion* = "*gini*", *max_depth* = *None*.
Logistic Regression	`linear_model.LogisticRegression` with default parameters; penalty parameter $C = 1$, *solver* = "*liblinear*".
Logistic Regression (SAG)	`linear_model.LogisticRegression` with default parameters, but with *solver* = "*sag*", which uses a Stochastic Average Gradient descent solver, this is supposed to be faster for large data sets.

5.1.4 Technical specifications

The technical specifications of the machine on which the evaluations were performed are listed below.

CPU Intel Xeon E5-1620 v3, Quad Core, 3.5GHz, 10MB

Memory 32 GB DDR4 ECC

Hard drive 60 GB SSD

5.2 Results

In this section we present the results from the experiments that were previously described. The first part of this section details the results of the UNSPSC segments experiments, and the second part the results of the experiments conducted on the hierarchical classifiers.

5.2.1 UNSPSC segments results

The weighted F_1-score of the models trained in the UNSPSC segments experiments can be seen in Table 5.2, and their training times are listed in Table 5.3. Both tables show a comparison of using and not using the features which were introduced in Section 4.2.3, referred to as *Extended Features* and *Basic Features* respectively. There are a few immediate things to notice while looking at these tables;

- The extended features that were extracted from the product titles and their GTIN did not significantly increase the score of the classifiers.
- There was a slight decrease in the weighted F_1-score of the trained models when trained using inverse class frequency balancing.
- There was no significant increase in F_1-score for SVM models trained using the Crammer-Singer algorithm, their training time however, increased greatly.
- The Naive Bayes models were by far the worst performing models, but were also the fastest to train by a great margin.
- Using the Stochastic Average Gradient descent solver for training the logistic regression proved to yield a large decrease in training time.

Only two of the confusion matrices that were generated are shown in this section, namely Figure 5.2 and Figure 5.3, to illustrate how the balancing parameter affected the models trained by the SVM learning algorithm. Specifically, it can be seen that less misclassifications into the majority class 43 are made, which was the general theme for all of the models that could use this balancing parameter. This is why the models trained in the hierarchical classifier experiments were trained using this parameter. The majority of the confusion matrices have been relegated to the appendix if further inspection by the reader is desired.

Table 5.2: Weighted F1-scores from the UNSPSC-segment experiments

Learning Algorithm	Weighted F1	
	Basic Features	Extended Features
Multinomial naive Bayes	0.85	0.85
Bernoulli naive Bayes	0.82	0.81
SVM	0.91	0.92
SVM*	0.91	0.91
SVM (CS)	0.91	0.91
SVM* (CS)	0.90	0.90
Decision Tree	0.92	0.92
Decision Tree*	0.91	0.91
Random Forest†	0.92	0.92
Random Forest*†	0.92	0.92
Logistic Regression†	0.92	0.92
Logistic Regression*†	0.91	0.89
Logistic Regression†(SAG)	0.92	0.92
Logistic Regression*†(SAG)	0.91	0.91

* Balanced, †Multi-core

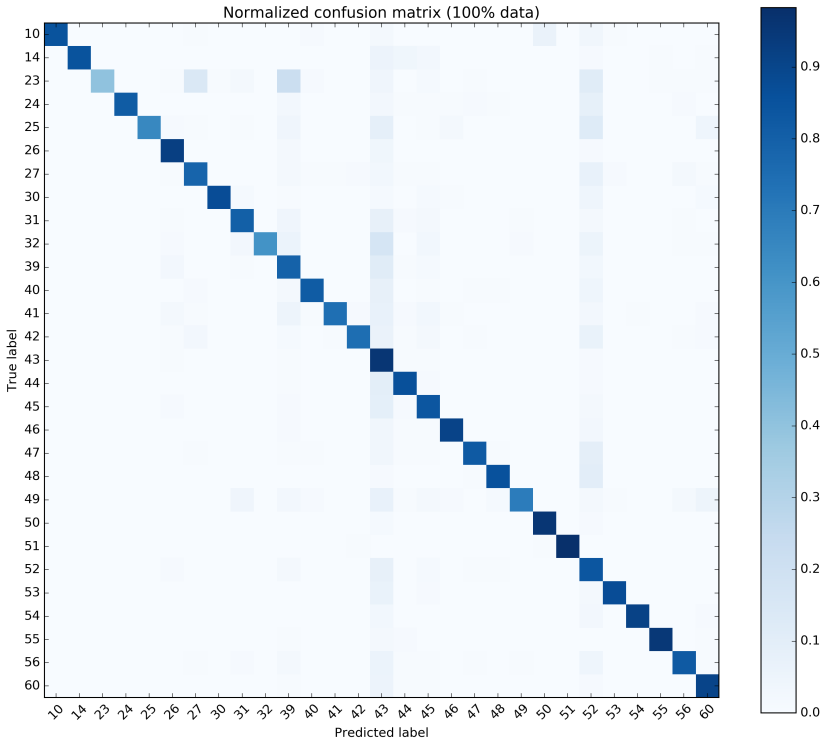

Figure 5.2: Confusion Matrix for Support Vector Machine model trained to predict UNSPSC segments using basic features.

Table 5.3: Training times in seconds from the UNSPSC segment experiments

Learning Algorithm	Basic Features (s)	Extended Features (s)
Multinomial naive Bayes	3	3
Bernoulli naive Bayes	3	4
SVM	669	901
SVM*	783	954
SVM (CS)	3,747	11,487
SVM* (CS)	2,324	16,659
Decision Tree	10,407	10,291
Decision Tree*	16,803	12,425
Random Forest†	7,845	9,282
Random Forest*†	6,502	9,659
Logistic Regression†	2,979	5,169
Logistic Regression*†	3,189	5,988
Logistic Regression†(SAG)	1,905	2,230
Logistic Regression*†(SAG)	1,871	2,231

* Balanced, †Multi-core

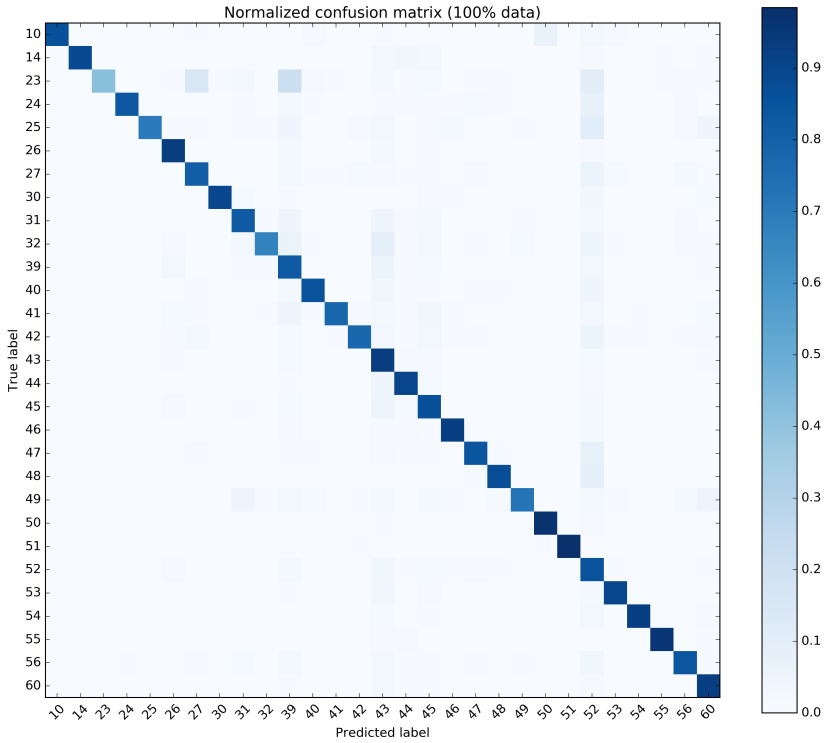

Figure 5.3: Confusion Matrix for Support Vector Machine model trained to predict UNSPSC segments using basic features and inverse class frequency balancing.

5.2.2 Hierarchical classifier results

The results from the hierarchical classifier experiments are summarised in Table 5.4, which lists the hierarchical F_1-score (hF_1), hierarchical precision (hP) and hierarchical recall

Table 5.4: Results from the hierarchical classifier experiments

Learning Algorithm	hF_1	hP	hR	Training Time (s)
Multinomial naive Bayes	0.77	0.74	0.81	923
SVM*	0.85	0.81	0.90	2,290
SVM*‡	0.85	0.81	0.90	1,247
Random Forest*†	0.84	0.80	0.89	74,974
Logistic Regression*†(SAG)	0.85	0.81	0.89	4,868
Logistic Regression*†‡ (SAG)	0.81	0.78	0.86	2,729

* Balanced, †Multi-core, ‡ L2-normalisation

(*hR*) of the trained models, as well as the training time of each of the learning algorithms. It is important to note that because of how the hierarchical classifiers are constructed, the training times listed in the table also include the time spent extracting features from the training set, which is why the Multinomial Naive Bayes model took an unexpectedly long time to train as can be seen when comparing the results from the two experimental setups. However, the time spent extracting features should be equal between the models, and when it was later measured, it totalled 246 seconds, which in theory can be subtracted from each of the training times.

5.3 Discussion

This section discusses the results from the two experiments as well as how the data used for them may have had an effect on the results.

5.3.1 UNSPSC-segments experiments

The purpose of this experiment was to see if classification using the available product data into UNSPSC categories was possible. The intention prior to the experiment was to try to collect more, richer data containing product attributes, which was expected to produce results with higher confidence. Unfortunately, we were unable to collect said data, making the results from this experiment more important.

Table 5.2 shows that a majority of the learning algorithms tested yielded an F_1 score above or equal to 0.90 which was surprisingly promising, and it proved that even with the expected dip in performance when adding consecutive classifiers in the hierarchy, we would be able to achieve a good result. The experiment also made it possible for us to limit the amount of algorithms to consider in the hierarchical classifier experiments which was useful since the hierarchical experiments were expected to take significantly longer to conduct.

Effect of extended features

To our great dissatisfaction, it can be seen from the results that there was no significant increase in F_1-score by extracting extra features using the feature extraction pipeline. Our results show that there are instances of learning algorithms that report both increased and

decreased scores between using basic and extended features, which to us seems slightly counter-intuitive as we expected there to be an universal increase or decrease in scores across the board.

To get to the bottom of which of the feature extraction steps of the pipeline are favourable and detrimental to the result, the steps would have to be tested in isolation, which is something that fell out of the scope of this thesis.

However, we strongly believe that the company prefix contained in the GTIN could be of great use in future product classification efforts, as companies often focus on producing a certain type of product. The fact that only 26% of the products had a GTIN may be part of why this feature did not have a large effect on the result.

Effect of inverse class frequency balancing

The slight drop in weighted F_1-score when applying balancing is expected. When digging deep into our results, we could see that the inverse class frequency balancing performed during training usually decreased the recall rate and increased the precision rate of frequent classes, while doing the opposite to infrequent classes.

As models are generally worse at making decisions for classes with few samples, the precision rate for these classes would often decrease more than their recall rate would increase, resulting in a decreased F_1-score for these classes. The F_1-score of frequent classes remained the same to a large extent, and because the weighted F_1-score tends to favour the scores of frequent classes, the models final weighted F_1-score largely remained the same.

We deemed that a slight increase in recall rate for the many infrequent classes at the expense of slight decrease in recall rate for the few most frequent was worth it, which is why we kept the balancing parameter in our following experiments. However, exactly what is the best setting is debatable.

In the end, looking at Figures 5.2 and 5.3, and indeed the ones in the appendix (Figures 3 to 10), there is obviously a drop in samples that get misclassified to the majority class: 43 and an increase in the recall of the minor classes.

Training time

The training times of the different learning algorithms are important factors to consider when deciding which method to use for training the hierarchical classifier. While many factors can be tweaked in order to decrease the training time of the Decision Tree and Random Forest models, this would come at the expense of prediction score. At the same time, it is important that the classifier is as accurate as possible, and a suitable balance between these two factors is therefore important.

There are a few learning algorithms that took a lot more time than the rest as well as not having notable classification performance: SVM with Crammer-Singer, Decision Tree, Random Forest and Logistic Regression. Consequently these were cut from the hierarchical classification experiment, except for Random Forest to prove how unsuitable it is for the hierarchical classification purpose.

How unsuitable these models are is especially obvious when considering that all of them except SVM with Crammer-Singer used four cores during training. Although the

functionality has not yet been implemented, it would be possible to train multiple models of the hierarchical classifier in parallel, which makes the learning algorithms that perform well on a single core look even more attractive.

Table 5.3 seems to suggest that there was a slight increase in training time when models were trained using inverse class frequency balancing, although exactly why this is the case is to us unclear. It should also be noted that some of the learning algorithms reported the opposite, which indicates that more experiments need to be conducted in order to be able to draw any definitive conclusion regarding this.

Problems with the experimental setup

These experiments fulfill their purpose by showing the viability of product classification using only product name and GTIN, but there are downsides.

The experiments focus on having a big set of learning algorithms and varying the variables common to them. The size of this set affects the time it takes to test a single variable, limiting the amount of variables that the experiment can consider within the given time frame of this thesis project. As such, it is probable that it would be possible to achieve even better results in terms of score and a higher confidence in the score's accuracy, by tweaking parameters such as regularisation terms.

The addition of cross-validation could possibly make it safe to conclude things in more detail about the effect of the extended features but it would also change the time it takes to run the experiments, which is now one day, to something significantly longer.

5.3.2 Hierarchical classifier experiments

As the segment experiments answered a lot of questions this experiment mostly serves to judge the quality of the hierarchical classifier and determine which learning algorithm was best suited for the task. There did not seem to be anything that would contradict the results in the previous experiment.

The best learning algorithm was found to be Support Vector Machines trained on L2-normalised input vectors, with an hF_1-score of 0.85 and a training time of 1,247 seconds. Below the reader will find a general discussion about the scores and training times.

Classification Scores

The results of this experiment was surprisingly positive. This experiment considered over 800 classes compared to the 31 that were considered in the segment experiment, and saw an hF_1-score 0.85 for the best models. What this score means is in essence that products were on average classified into 85% of their hierarchical classes, the number of which ranged from one to four, from UNSPSC segment to commodity.

Consistent with the results from the UNSPSC experiments, Table 5.4 show SVM, Random Forest and Logistic Regression to be very close in terms of classification score and that Multinomial naive Bayes does not really manage to compete with these algorithms in terms of score, which is not really unexpected.

As expected, the scores of the different learning algorithms did not change. Since the hierarchical classifier does not try to exploit anything unique to the individual learning

algorithms, there should not be any difference.

It can also be seen that normalisation of the input vectors did not have any significant effect on the score when used to train the SVM-classifier, but a large decrease when used together with logistic regression.

Training Time

Our expectation was that the training time for the hierarchical experiments would be significantly higher than in their counterparts in the segment experiments. Where each hierarchical experiment has a root model that very much resemble the model trained in the corresponding segment experiment, but with the addition of a lot of smaller models to train. These smaller models only consider a small part of the training set and so they can be expected to train faster then the root.

This increase in training time was confirmed for all learning algorithms, of which SVM and logistic regression saw an increase of approximately 2,5 times. Multinomial naive Bayes, saw a much greater increase, which is not surprising considering its short training time in the initial experiments. This increase suggests that there is a large overhead when training naive Bayes models of any size. The training time for the Random Forest model however, show that much tweaking of the training parameters has to be done for the learning algorithm to even be considered an option.

Another noteworthy thing with regards to the training time, is the expected reduction in training time when applying normalisation. Applying L2-normalisation before training a model using SVM yielded a reduction in training time of 1,043 seconds (46%), a huge improvement. The same could be seen for the model trained with logistic regression, but with a decreased score, which suggests that this should not be used.

Problems with the experimental setup

This experimental setup suffers from the same problem as the segment experiments does, but to a lesser extent, namely that more focus should have been on experiments with a specific learning algorithm, and tweaking its learning parameters.

In this experiment the hierarchical hF_1 score is used in comparison to the non-hierarchical weighted F_1, which makes it a bit difficult to compare the results of the experiments, although there is no definitive solution to this problem, because the performance simply cannot be measured in the same way.

5.3.3 Data

There are several problem areas with the data which are discussed here.

Imbalance

When talking about the distribution of products across the categories of the UNSPSC taxonomy, it can be assumed to be imbalanced to some degree for a number of reasons. One reason is that different market segments produce differing amounts of products, as different segments are more volatile. In some market segments, products may be relevant

for very short periods of time, while in other segments a product can exist and be sold for several years without becoming outdated. As a result, more products will exist in some parts of the taxonomy.

Another reason for imbalance is that the UNSPSC taxonomy itself is imbalanced. When a part of the taxonomy is updated, it will typically see an increase in number of categories. Furthermore, in some industries, it is more important to be specific, for example in the health care industry. This segment in the UNSPSC taxonomy contain many more categories than most others.

Unfortunately it is unlikely that the distribution of products in the training data reflects these real life imbalances, with parts of it like the data from *Icecat* and *Open Food Facts* containing mostly electronics and food respectively. This dissonance is potentially harmful to the predictive quality of the models that are trained on the data. In our case, the classifier would possibly favour categories that in reality do not have many products or ignore categories that do.

We try to combat this by using inverse class frequency balancing, but apart from that, there is not much that can be done to prevent this except collecting more data.

Introduction of errors in data

During the collection of the data used in the experiments, it is possible that some errors were introduced. Firstly, the data sets can be assumed to inherently contain some small errors since they are big, which is especially the case for the data from Open Food Facts as it is an open collaborative project.

The refinement of the class names may also potentially have corrupted some samples, and finally, errors could have been introduced during the translation of the class names to UNSPSC, as it was done manually by human beings, and as there may not be a distinct one-to-one mapping between the two sets of categories that were translated.

Is is unclear how big of a problem this really is, as a lot of these errors are introduced through human interaction and consequently not easily analysed.

Incomplete taxonomy coverage

With a taxonomy as big and seemingly ever-expanding as the UNSPSC, it is hard to get data that cover all categories. In our case we managed to collect data containing products from 1,628 different categories, but in the end, only 829 of these were represented to the degree that we were confident in using them as output categories in our models. This of course can not represent the over 60,000 classes that are in the version of UNSPSC that we currently use, which is a problem since these of course would never be predicted by our classifier.

A big part of the taxonomy will presumably be uninteresting to end consumers, and as a result, the size of the UNSPSC taxonomy could be reduced to roughly 45,000 classes by ignoring some segments such as raw materials and services (which are not products). Unfortunately this still leaves a lot of categories that will never get predicted in the classifiers current state.

Chapter 6
Conclusions

This thesis describes the implementation of a hierarchical product classifier using python and scikit-learn, and how it could be used together with an impact factor database to compute the socio-ecological impact of a product. The hierarchical classifier has been evaluated using a number of different learning methods, and our results suggest that the optimal learning method for training such a model is Support Vector Machines using inverse frequency class balancing.

By extracting unigram and bigram features from the brand and title of products and representing their binary counts as vector space models using feature hashing, a hierarchical classifier was trained on a data set of 2.5 million products using the aforementioned learning algorithm. This model was proven to be able to classify a test set of 600,000 products into 800 categories with a hierarchical F_1-score of 0.85, which is an acceptable score for its intended purpose.

The effect of feature extraction methods developed as part of this thesis project proved to be inconclusive, but it is the opinion of the authors that the company prefix contained in the GTIN's of products should be further investigated.

The result of this thesis may in the future be used at Meta Mind in order to estimate products socio-ecological impact on the environment, although more data, some finishing touches and also some surrounding infrastructure is necessary for it to be used in a real world setting.

6.1 Future work

There are many interesting aspects of this thesis that are left to explore, some of which fall out of the scope of this thesis, and they are described here.

6.1.1 Improved feature extraction pipeline

There are some known things that could possibly improve the feature extraction pipeline.

The idea of deriving predictive information from alphanumeric strings, such as "Xerox *700i*" can likely be improved. The current approach employed by most of the steps in the feature extraction pipeline is to simply apply a regular expression to product titles in order to extract more detailed features. The AlphanumericWords step of the pipeline for example, creates a binary feature out of any product brand or title that has a word that contains both digits and letters.

The intention of this is to make for example "Xerox *700i*" more similar to "Xerox *800i*" than to "Xerox Drivers", but it is probably far too general to make any actual difference, as it would think "Xerox *NOTAPRINTER123*" is as close as the first example. This can likely be improved by making more sophisticated regular expressions, or by appending tokens based on word forms to the product title, such as "Xerox 700i <3dig><1char>". Unigrams and bigrams could then be extracted in the same way as regular words, feature hashed to create a vector space model.

Another possible improvement to the pipeline would be to add a stemmed version of the text. That is to instead of either doing stemming or not, as previously investigated by Yu et al. (2012), to do both and concatenate the results. The effect of this on classification performance is something that we would like to investigate further in the future.

As previously mentioned, the features introduced in Section 4.2.3 did not get sufficiently tested. Testing these individually and identifying the problems there will hopefully improve classifiers that consider them.

6.1.2 Beam Search

When predicting a class, some learning algorithms such as logistic regression can return a probability distribution, a set that describes the probability of each class. In non-hierarchical classifiers it is obvious to pick the one which has the highest probability, as that class is most likely the correct one. In hierarchical classifiers on the other hand, these probability distributions make up a tree that can be searched using beam search to find the best class, while also considering the subtrees of the likely alternatives.

The beam search would be done by using some function to select a subset of the output classes based on their probabilities instead of picking the output class with the highest probability, which is indirectly done by our classifier today. The product being classified could then be classified using the models that represent the set of output classes. This would be repeated until the product had been classified as far as possible into the UNSPSC taxonomy upon which we would be left with a set of possible classification *paths* through the taxonomy tree. When this stage has been reached, a combined probability measure could be computed by multiplying the probability at each level of the classification process, upon which an output category could be selected based on some criteria.

6.1.3 Hierarchical Stopping Condition

One of the problems that our hierarchical classifier faces as it is today is that it always tries to categorise products into leaf categories of the UNSPSC taxonomy, or what can

be referred to as UNSPSC *commodities*. This is essentially the equivalent of seeing the taxonomy as complete, in that it takes every possible type of product into account to the smallest detail, which of course is not true.

As an example, take a smart phone screen protector, for which there is no leaf category. A product of this type should instead be classified as a phone accessory, a category that is not a leaf category. However, our classifier will always try to categorise the screen protector into one of the subcategories of phone accessories, even though it should not.

Aixin and Ee-Peng (2001) solved this for a hierarchical classifier made up of binary classifiers by employing an additional classifier at each node of the taxonomy tree. These extra classifiers are trained to decide if further classification into the subtree is performed or not. For a multi-class classifier these extra classifiers can simply be replaced with the addition of a new class, that contains all the products that should not get further classified.

6.1.4 Language Filter

Currently, our hierarchical classifier is trained to predict products with English titles only, and as such, products from many parts of the world cannot be classified. A solution to this would be to employ a language filter, which would be used to identify which language a product is described in.

This is not something that we have been investigating extensively, but it could be done by looking at stop words contained in product titles and comparing them to lists available in the Natural Language Toolkit (NLTK)(Bird, 2006).

6.1.5 Parallel computations

Something that would speed up the training time quite a bit for models trained with single-core learning algorithms such as Support Vector Machines is to be able to train multiple models of the hierarchical classifier at the same time. As the hierarchical classifier is trained from bottom up, and two sibling models in the tree use different sets of training data, they could easily be trained at the same time if there are no memory restrictions.

Parallel processing in Python is very much a possibility, which makes us believe that this is something that could have a great effect on training times.

6.1.6 Feature selection

Our hierarchical classifiers are as explained above made up of a multitude of individual classifiers, each trained using a large number of features. This combination makes the final hierarchical model very big, especially because of the extra large feature space required when using feature hashing. The extra large feature space is defined during feature hashing to avoid collisions between features, and as a consequence of this in combination with the short product titles, we believe that there are many columns in the output vector of this step that are not used.

Feature selection would solve this by removing features that are not seen to help the classification. Doing this for each of the classifiers could possibly yield much smaller

classifiers since the overlap in word usage between classifiers is probably not complete, not all words are used in relation to all types of products.

This would most likely add to the time it takes to train the hierarchical classifiers, but it is not certain, as it could be negated by the fact that models trained on smaller feature sets typically take a shorter time to train. Additionally, it would decrease the amount of memory and computing power needed by the server on which the classifier is expected to run.

6.1.7 Cross validation parameter tuning

Something that we have not done in our thesis, is to more finely tune the parameters of the trained models, for example the regularisation parameter C for the hierarchical classifiers trained using Support Vector Machines.

This could be done using the built-in features of scikit-learn and should preferably be done for each individual model that make up the hierarchical classifier combined with cross validation. As such, this would increase the training time significantly of the hierarchical classifier, but as it is relatively short as of now, this may not be a problem.

Being able to store and import this information separately before training would make it possible to not have to fine tune these parameters between every training session for models where the training data has not changed significantly, which would also be helpful.

6.1.8 Combination with web scraping

We would very much like to combine this classifier with a web scraper. A web scraper is simply a bot that navigates websites and extracts information along the way according to some algorithm. As we in this thesis proved that product classification into large sets of classes can be performed with a high success rate based primarily on product titles, it would be interesting to combine our classifier with an extended version of the web scraper described by Talaika et al. (2015). Their scraper focuses on extracting names from web pages containing unique identifiers, such as GTIN's and product titles.

Although we were not able to conclude that GTIN's can actually be used to increase the performance of our classifier, it is still possible that it could. This makes this approach very interesting, and could be used in conjunction with the impact factor database described earlier to create a database of products and their environmental impact, which is something that we definitely will look into in the future.

Bibliography

Abels, S. and Hahn, A. (2006). Reclassification of electronic product catalogs: The "apricot" approach and its evaluation results. *Informing Science*, 9:31 – 47.

Aixin, S. and Ee-Peng, L. (2001). Hierarchical text classification and evaluation. *Proceedings 2001 IEEE International Conference on Data Mining*, page 521. ISBN: 9780769511191.

Bikel, D. M., Schwartz, R., and Weischedel, R. M. (1999). An algorithm that learns what's in a name. *Machine learning*, 34(1-3):211–231.

Bird, S. (2006). Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167.

Crammer, K. and Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292.

Ding, Y., Korotkiy, M., Omelayenko, B., Kartseva, V., Zykov, V., Klein, M. C., Schulten, E., and Fensel, D. (2002). Goldenbullet in a nutshell. In *FLAIRS-2002: The 15th International FLAIRS Conference*, pages 403–407.

Grenada Research (2001). Why coding and classifying products is critical to success in electronic commerce. `https://www.unspsc.org/Portals/3/Documents/Why%20Coding%20and%20Classifying%20Products%20is%20Critical%20to%20Success%20in%20Electronic%20Commerce%20(October%202001).doc`. Accessed 2016-01-18.

GS1 (2016). Gs1 company prefix length. `http://www.gs1.org/gcp-length`. Accessed 2016-04-26.

GS1 (2016). GS1 general specifications. http://www.gs1.org/sites/default/files/docs/barcodes/GS1_General_Specifications.pdf. Accessed 2016-01-18.

Huynh, D. (2013). the future of the refine project. https://groups.google.com/forum/#!topic/openrefine/a3R6afKb4-4. Accessed 2016-05-11.

Icecat (2016). About icecat. http://icecat.co.uk/en/menu/about/index.html. Accessed 2016-05-11.

Kim, Y.-g., Lee, T., Chun, J., and Lee, S.-g. (2006). Modified naïve bayes classifier for e-catalog classification. In *Data Engineering Issues in E-Commerce and Services: Second International Workshop, DEECS 2006, San Francisco, CA, USA, June 26, 2006. Proceedings*, pages 246–257.

Kiritchenko, S., Matwin, S., Nock, R., and Famili, A. F. (2006). Learning and evaluation in the presence of class hierarchies: Application to text categorization. In *Advances in Artificial Intelligence: 19th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2006, Québec City, Québec, Canada, June 7-9, 2006. Proceedings*, pages 395–406, Berlin, Heidelberg. Springer Berlin Heidelberg. ISBN: 978-3-540-34630-2.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397.

Manning, C. D., Raghavan, P., Schütze, H., et al. (2008). *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge.

McDonough, W. and Braungart, M. (2002). *Cradle to Cradle: Remaking the Way We Make Things*. North Point Press.

McKinney, W. and PyData Development Team (2016). pandas: powerful Python data analysis toolkit. http://pandas.pydata.org/pandas-docs/version/0.18.0/pandas.pdf. Accessed 2016-04-20.

Miller, R. E. and Blair, P. D. (2009). *Input-Output Analysis: Foundations and Extensions*. Cambridge university press.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Russel, S. J. and Norvig, P. (2014). *Artificial Intelligence: A Modern Approach (3rd ed.)*, pages 109,706–707,848,850. Prentice Hall. ISBN: 978-1-292-02420-2.

Scikit-learn (2014). Model evaluation. http://scikit-learn.org/stable/modules/model_evaluation.html. Accessed 2016-05-18.

Talaika, A., Biega, J., Amarilli, A., and Suchanek, F. M. (2015). Harvesting entities from the web using unique identifiers - IBEX. *CoRR*, abs/1505.00841.

Wamai Egesa, A. (2016). Analysis of financial transactions using machine learning. ISSN: 1650-2884.

Wolin, B. (2002). Automatic classification in product catalogs. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, pages 351–352.

Yu, H.-F., Ho, C.-H., Arunachalam, P., Somaiya, M., and Lin, C.-J. (2012). Product title classification versus text classification. *Technical Report*. Accessed 2016-04-20 `https://www.csie.ntu.edu.tw/~cjlin/papers/title.pdf`.

Yu, H.-F., Ho, C.-H., Juan, Y.-C., and Lin, C.-J. (2013). Libshorttext: A library for short-text classification and analysis. *Rapport interne, Department of Computer Science, National Taiwan University. Software available at http://www. csie. ntu. edu. tw/˜ cjlin/libshorttext*. Accessed 2016-04-20.

Appendices

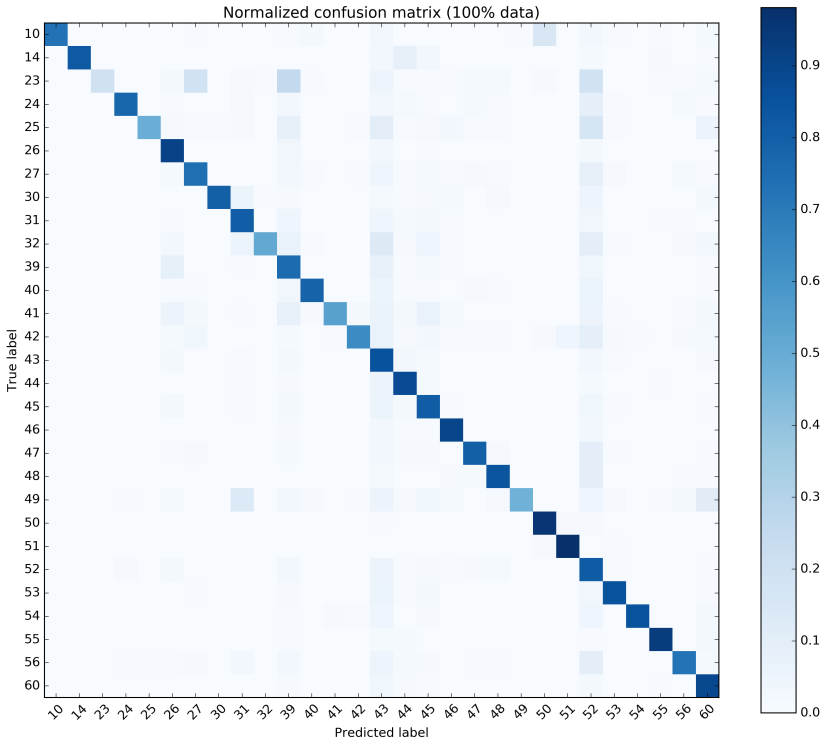

Figure 1: Confusion Matrix for Multinomial Naive Bayes model trained to predict UNSPSC segments.

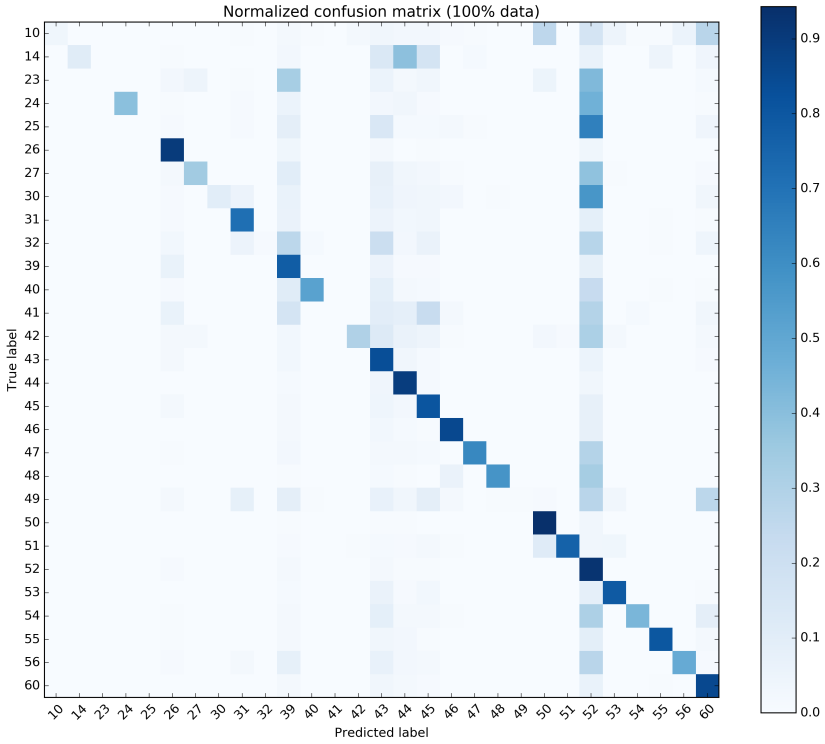

Figure 2: Confusion Matrix for Bernoulli Naive Bayes model trained to predict UNSPSC segments.

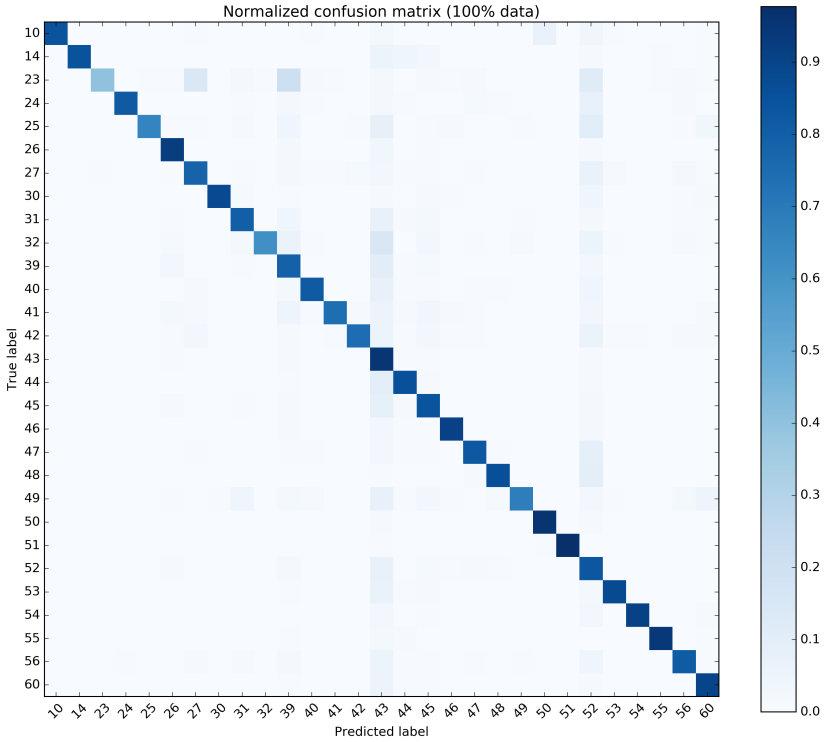

Figure 3: Confusion Matrix for SVM model trained using the Crammer-Singer algorithm to predict UNSPSC segments.

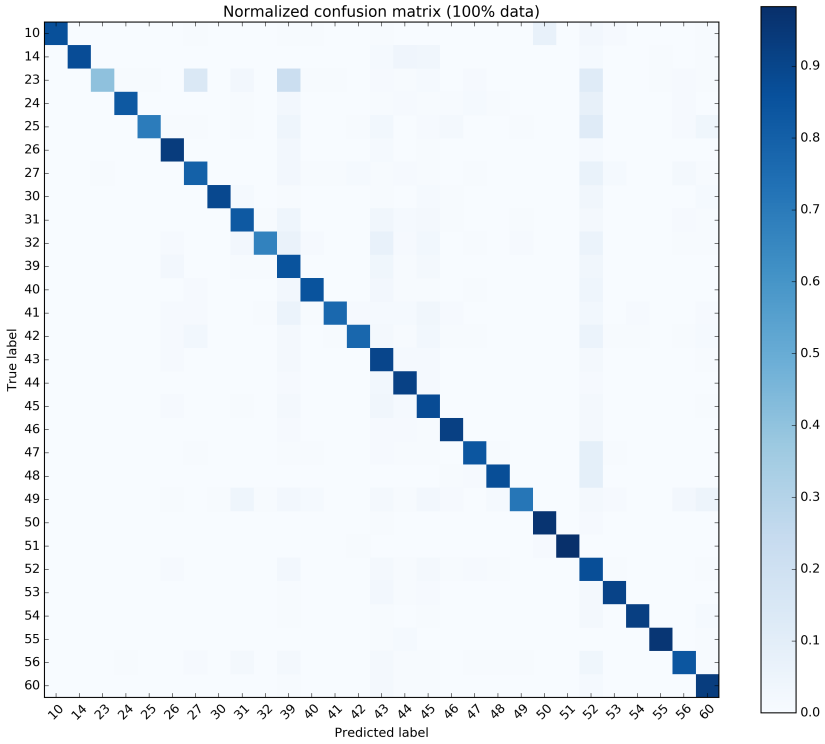

Figure 4: Confusion Matrix for SVM model trained using the Crammer-Singer algorithm and inverse class frequency balancing to predict UNSPSC segments.

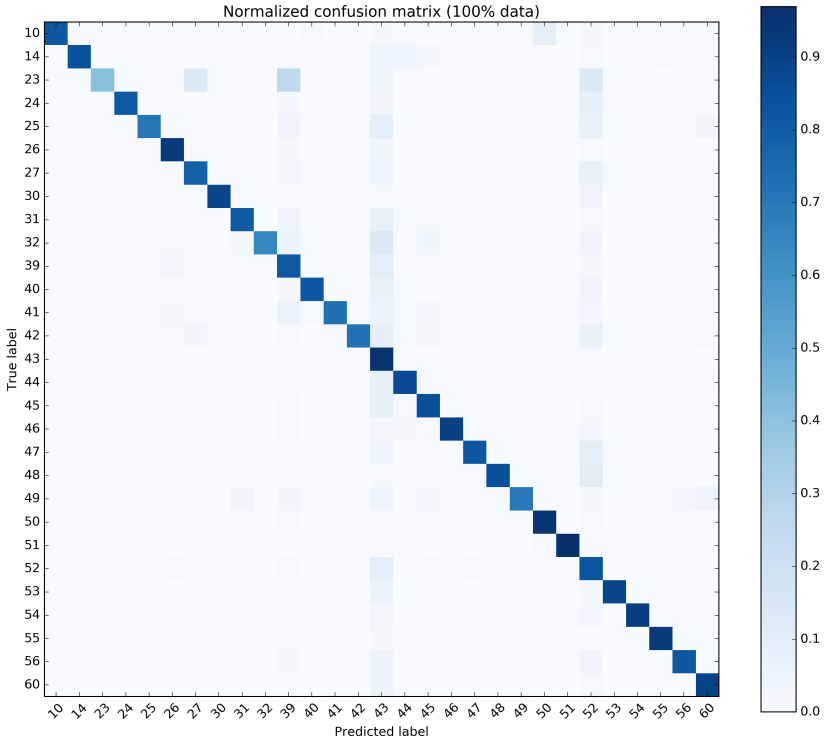

Figure 5: Confusion Matrix for Decision Tree model trained to predict UNSPSC segments.

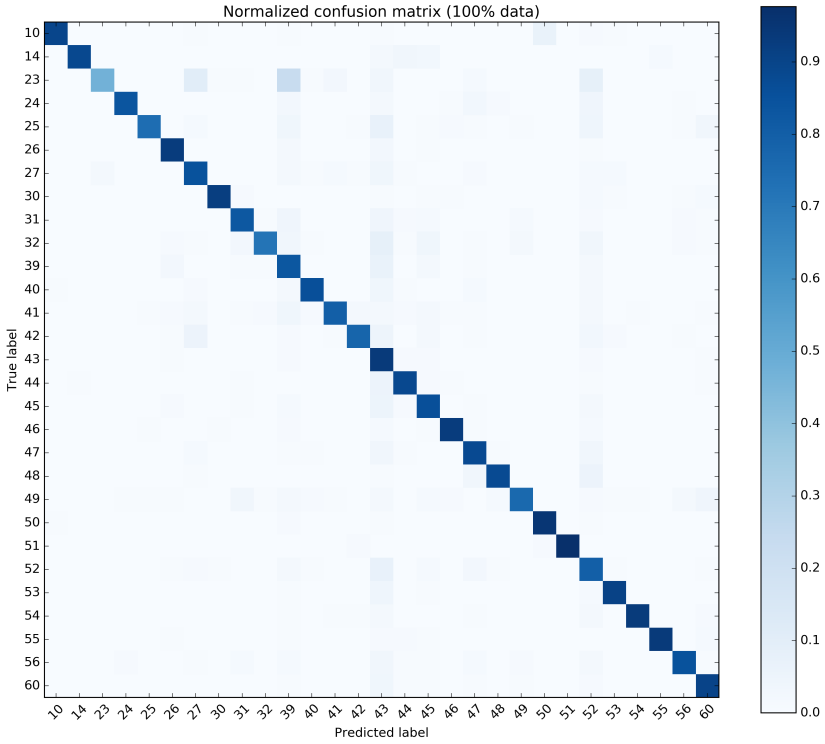

Figure 6: Confusion Matrix for Decision Tree model trained using inverse class frequency balancing to predict UNSPSC segments.

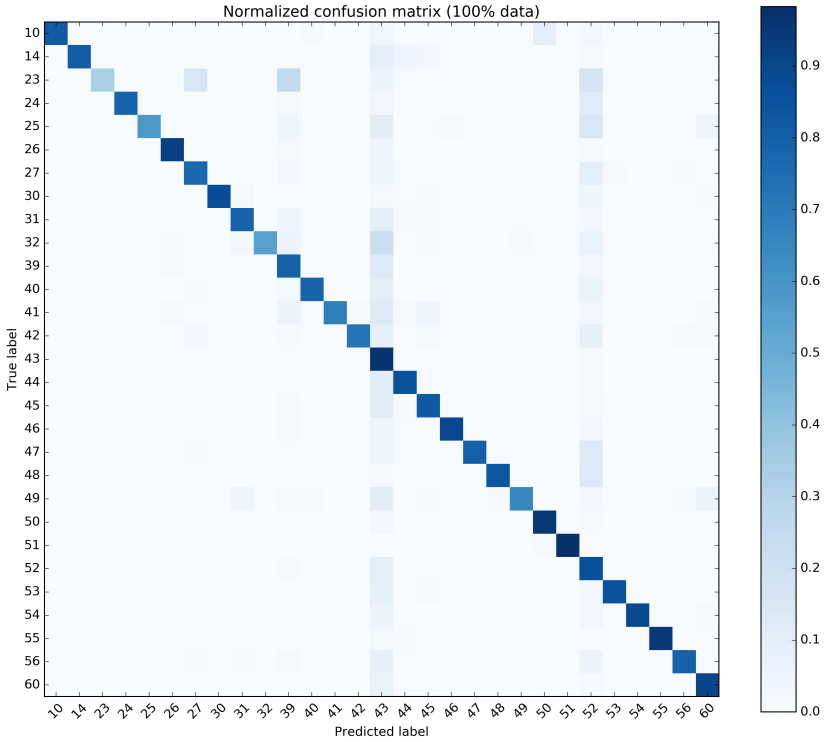

Figure 7: Confusion Matrix for Logistic Regression model trained to predict UNSPSC segments.

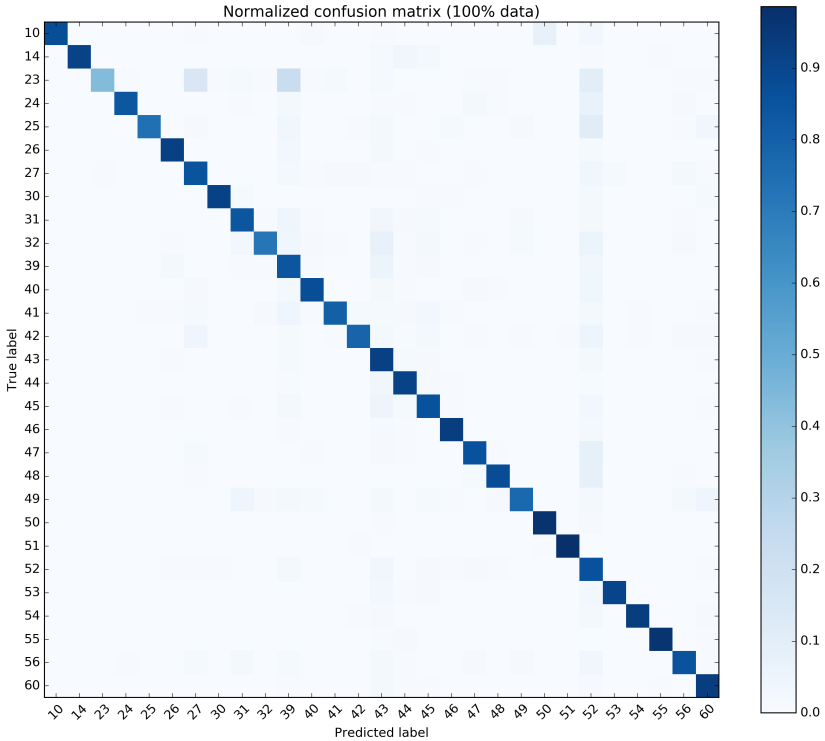

Figure 8: Confusion Matrix for Logistic Regression model trained using inverse class frequency balancing to predict UNSPSC segments.

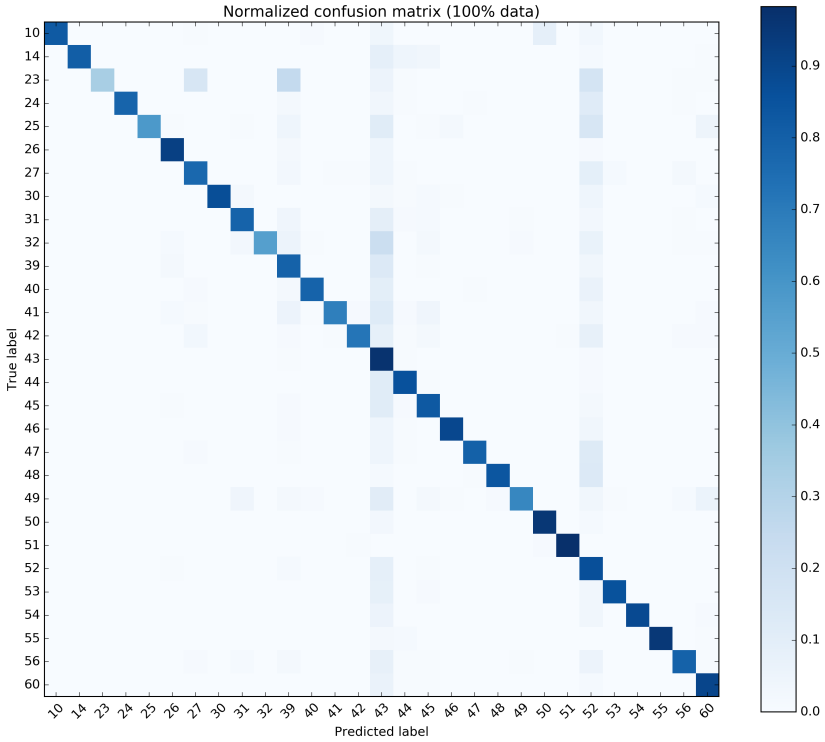

Figure 9: Confusion Matrix for Logistic Regression model trained using a SAG-solver to predict UNSPSC segments.

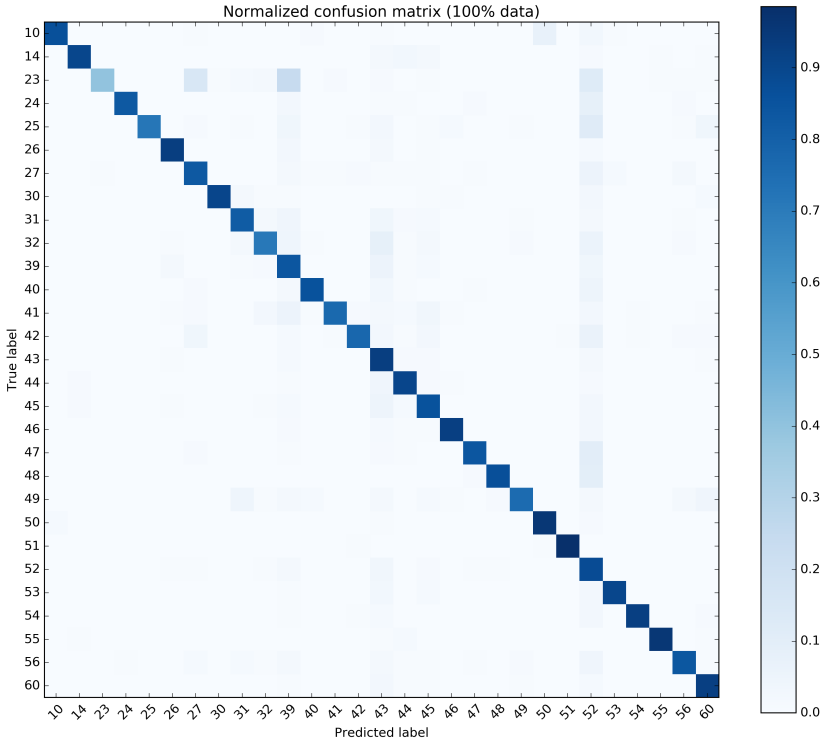

Figure 10: Confusion Matrix for Logistic Regression model trained using a SAG-solver and inverse class frequency balancing to predict UNSPSC segments.

Figure 11: Graph showing class distribution over all UNSPSC levels, where center circle is root, and each layer outside of it refers to an UNSPSC level, from Segment to Commodity. Omitting class names to simplify visualisation

$$weightedF_\beta = \frac{1}{\sum_{l \in L} \mid \hat{y}_t \mid} \sum_{l \in L} \mid y_t \mid F_\beta(y_l, \hat{y}_l)$$

Figure 12: Weighted F function as defined by (Scikit-learn, 2014)

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTATIONSDAG 2016-06-03

EXAMENSARBETE Product Classification - A Hierarchical Approach
STUDENT Anton Karlstedt, Mikael Karlsson
HANDLEDARE Pierre Nugues (LTH), Kristian Rönn (Meta Mind AB)
EXAMINATOR Elin Anna Topp (LTH)

Hur påverkar en produkt miljön?

POPULÄRVETENSKAPLIG SAMMANFATTNING **Anton Karlstedt, Mikael Karlsson**

Med stora miljöutmaningar framför oss gör artificiell intelligens det möjligt för konsumenter och företag att kunna beräkna och jämföra olika produkters sociala och ekologiska fotavtryck.

Att varje handling har en konsekvens är knappast någon nyhet. För de allra flesta, åtminstone för oss själva, är det en självklarhet att den där köttfärsen kommer att få fredagens tacokväll att bli en riktig höjdare likaså att sommarens värmebölja kommer likna en dröm i den nya luftiga skjortan. Konsekvenserna av att den stackars kossan ska födas upp och till slut transporteras till närmsta kyldisk däremot, dem är det få som faktiskt är medvetna om och detsamma gäller skjortan. Hur påverkas egentligen människor och vattendrag på andra sidan klotet av de gifter som används för att vi alla ska slippa lida igenom den *olidliga* hetta som är så karaktäristisk för den svenska sommaren?

Denna vetskap kan naturligtvis framstå som skrämmande men den är samtidigt väldigt viktig, framförallt för oss som utgör den del av jordens befolkning som är allra rikast. Vår överkonsumtion påverkar miljön något enormt över hela världen men det behöver inte vara så, för vi har stora möjligheter både ekonomiskt och teknologiskt att förändra hur vi påverkar vår omgivning genom att anpassa våra livsstilar och konsumtionmönster.

Vårt examensarbete gör det möjligt att beräkna och som följd jämföra hur mycket särskilda produkter påverkar miljön på olika sätt. Vi hoppas att detta i framtiden kommer att kunna användas för att låta individer och företag göra smarta val – val baserade på fakta och val för att minimera sin miljöpåverkan.

I vårt examensarbete bygger vi ut ett system utvecklat av företaget Meta Mind AB. De har samlat resultaten av vetenskaplig forskning världen över för att kunna beskriva hur olika slags produkter påverkar miljön. Exempelvis kan systemet säga hur mycket växthusgaser som släpps ut av att köpa låt oss säga getost till ett värde av en dollar i Sverige. Det enda problemet är att vi på något sätt måste veta att en viss produkt är just getost och ingenting annat. Det finns många olika slags ostar av olika fabrikat, tillverkade på olika sätt och med olika ingredienser: Arlas prästost, Zetas parmesanost, Icas laktosfria halloumi. Hur kan vi veta vad som är getost eller inte utan att explicit behöva ange vad varje produkt i hela världen är för någonting?

Genom att använda oss av artificiell intelligens och speciellt en teknik som kallas maskininlärning har vi utvecklat ett datorprogram som kan lära sig att kategorisera produkter baserat på deras fabrikat och titel. På så sätt kan vi utnyttja den insamlade forskningsdatan för att beräkna hur mycket en produkt påverkar miljön.

I våra experiment, där programmet fått lära sig genom att titta på 2.5 miljoner produkter, har det lyckats kategorisera produkter i 800 kategorier med 85% säkerhet. Framöver hoppas vi kunna ta del av mer produktinformation för att kunna klassificera och beräkna miljöpåverkan av ännu fler slags produkter.