Discussion paper

A comprehensive view of machine learning techniques for CPI production

Alexander Harms
Siemen Spinder

November 2019

Content

Summary

The Consumer Price Index (CPI) is a statistic that measures the price changes of goods and services. For its calculation, Statistics Netherlands divides all goods and services over a set of predetermined categories and takes a weighted average of price changes within the categories. Putting the right goods in the right categories is currently done by means of link tables and rule sets, that use the vendor's division of the products and the product information to determine the correct category. In this paper, we present machine learning approaches that automatically classify products into categories by means of the products' description, ID and brand. We also investigate using image data as a feature.

Traditional machine learning approaches to classification, i.e., Logistic Regression, Naive Bayes, Random Forest and Support Vector Machines are compared with a boosting method, XGBoost and the deep learning methods Recurrent Neural Networks (RNNs), Attention Layers and Convolutional Neural Networks (CNNs). In addition, we experiment with semi-supervised learning; using unlabeled data to increase the accuracy of our classifier.

The best model for product classification is found to be Logistic Regression, which offers an accuracy of 92.14% with a computation time of 43 seconds. The semi-supervised techniques incremental learning and co-learning are identified as ways to boost performance of supervised techniques.

When image data is not considered, deep learning approaches do not provide any benefit to product classification, as product descriptions are too short and non-cohesive. On the other hand, CNNs can provide value as feature extractors for image data. These image features make the final model perform better, but involve web scraping for a considerable amount of time to obtain the images.

Acknowledgements

We would like to thank Florin Barb and Guido van den Heuvel for their guidance and their input during this project and we would like to thank Els Hoogteijling for the opportunity to carry out this project as a part of her team. In addition we would like to thank Guido van den Heuvel for reviewing this discussion paper on November 1st, 2019.

Keywords

Consumer Price Index (CPI), multiclass classification, deep learning, XGBoost, semi-supervised learning

1. Introduction

Distributing thousands of products over categories is a challenging task, that needs to be carried out to calculate the Consumer Price Index (CPI). Currently this is done by means of link tables between the vendor's own division of the products and the categories used in the production of the CPI. Additionally rule sets describing how the product information maps to the categories are used. Making link tables and rule sets is a time consuming task that can be replaced by algorithms that learn the mappings themselves. In this paper we explore various of such *machine learning* algorithms to exploit information in the brand, ID, description and image of products in order to determine the correct category for the product.

The Consumer Price Index (CPI) is calculated by taking a weighted average of the price changes of goods and services, where the weights are determined by the total revenue from consumer sales of a product category. Before it can be calculated, products of different vendors need to be divided in product categories, the COICOP (Classification of Individual Consumption by Purpose) categories, that are captured at a European level in the ECOICOP (European COICOP) standard.

To prevent needing to manually label all products in a COICOP-category, Statistics Netherlands leans on information provided by the vendor in its transaction data or on its website. The information is linked to the COICOP categories by means of link tables and rule sets. Making these is time consuming and needs to be repeated many times, because products and product codes differ between vendors and differ in time.

Therefore, we propose to use machine learning (ML) to automatically classify products. By means of machine learning, only a subset of the data has to be labelled, after which the model can be used to label the rest of the data and perhaps future datasets. Besides that, the model may carry over between vendors, making a more general approach possible, rather than needing a specific approach for every vendor.

In this paper we explore various ways to classify products, in particular items of clothing of one vendor. Each time we use a product’s ID, description and brand as input. These are different kinds of inputs: structured (brand), semi-structured (product ID) and unstructured (product description). This requires us to explore ways to combine such types of data. The classification target that is used is not the ECOICOP itself as manually labelled data was not yet available, but rather a proxy, namely the category the vendor lists on its website. Examples of these categories are ‘tops’ and ‘low shoes’.

To classify CPI goods, several kinds of supervised ML-models are explored. Supervised methods differ from unsupervised methods as they assume the data is labelled and parameters are learned using the labelled data. The traditional supervised approaches we use are Naive Bayes (NB), Random Forest (RF), Support Vector Machines (SVM) and Logistic Regression (LR). These have been implemented efficiently in the Python package scikit-learn (Pedregosa, et al., 2011). As an addition to the traditional approaches we use XGBoost (Chen & Guestrin, 2016), a boosting technique, that is based on iteratively updating a decision tree.

On top of these models, we look at deep learning approaches. Deep learning refers to the use of multi-layered neural networks. It does not refer to one particular model, but many types of heavily parameterized architectures to perform tasks. Deep learning approaches have shown stellar performance in fields such as image recognition (Krizhevsky, Sutskever, & Hinton, 2012), sequence-to-sequence translation (Bahdanau, Cho, & Bengio, 2014) and text classification (Yang, et al., 2016).

Examples of neural network architectures are feed forward neural networks (FF-NN), recurrent neural networks (RNN) and convolutional neural networks (CNN). In each of these an input is taken, linearly transformed by means of weights and followed by a non-linear activation function. This can be repeated many times, in different ways, and with different inputs. For example, in a feed forward network, every input has its own weight, resulting in a large amount of weights. On the contrast, convolutional neural networks employ the principle of shared weights. A 'filter' of a certain size is run over the input, each time applying the same weights to every item captured in the input window. This heavily reduces the amount of parameters. Later on in this paper, we will in more detail describe the inner workings of some architectures.

In addition to improving performance by means of neural networks we wish to reduce the amount of hand labeled products necessary to get a sufficiently high accuracy. To do this, we employ the large amount of unlabeled data that is present in both transaction data and web scraped product data. This branch of machine learning, in which both labeled and unlabeled data is used, is referred to as semi-supervised learning. Techniques that will be discussed are incremental learning, tri-training (Zhou & Li, 2005; Ruder & Plank, 2018) and co-learning (Zhou & Goldman, 2004).

2. Data

2.1 Data description

We make use of web scraper data that is used for the calculation of the Consumer Price Index (CPI). In particular, we use data from a single clothes webshop. This dataset will be referred to as *K* (referring to the Dutch word for clothes, *'kleding'*). The data was gathered by Statistics Netherlands and is only available within the CPI production environment. The text in this dataset is in Dutch. The data contains three types of variables:

- Categories (e.g. brand),
- Identification numbers (product ID)
- Text data (product descriptions)

These three types need to be handled in different ways.

2.2 Data transformations

All variables are transformed to something that can be used in a mathematical model, i.e. numbers. When carrying out this transformation, it is important that as much relevant information as possible is taken into account, so the machine learning model can use this information for classification. Every column from the data set is separately transformed and the results of these transformations are subsequently concatenated. The numeric variables that result from processing the data are referred to as 'features'.

2.2.1 Categories

The most important information that a category contributes to the machine learning model is whether the product does or does not fit in a certain category. Therefore, for every category a dummy variable is made with 1 representing the case where a product does fit in a certain category and 0 the case where a product does not fit. This is called 'One Hot Encoding'.

2.2.2 Identification numbers (IDs)

Frequently the identification numbers reflect a certain product subdivision that a vendor has made. Knowledge of the structure of IDs is therefore favorable to product classification, as products that share parts of their ID often belong to the same category. Therefore, we split identification numbers and feed them to the machine learning model in pieces.

For each consecutive string of a certain length a dummy variable is made, a variable that is 1 when the string is present and 0 when it is not. If the length of three is chosen and the ID of interest is ‘M35Z4B’ this means inserting dummy variables for the strings ‘M35’, ‘35Z’, ‘5Z4’ and ‘Z4B’. Rather than splitting the product ID up in strings of one length, it is also possible to split the ID in several lengths, for example in groups of 3 and 4 characters. In this example the ID ‘M35Z4B’ is represented by not only the strings ‘M35’, ‘35Z’, ‘5Z4’ and ‘Z4B’, but also the strings 'M35Z', '35Z4' and '5Z4B'. The technique of splitting up in consecutive strings is referred to as ‘character N-grams’ where *N* refers to the size of the character-combinations. From informal tests it appeared that a value of N=2 gave the best results for the character N-grams transformation of the ID string.

In an ideal situation one would study the structure of the product ID to determine the best way to split the ID code. For example, say that the product ID ‘M35Z4B’ is built on the principle that the first letter 'M' is the gender of the target demographic, the next four characters '35Z4' is a product code and the last letter 'B' indicates the color of the item. The ideal way to represent 'M35Z4B' in dummy variables would be to have a dummy variable for 'M', '35Z4' and 'B'. Unfortunately product IDs can differ between companies, so such a transformation of the product ID based on prior knowledge of the ID structure cannot be applied for every vendor of which you want to categorize products without repeating this analysis. Therefore, we apply the character N-grams transformation and let the model figure out which combination of letters and/or digits are the most meaningful.

2.2.3 Text data

There are two types of methods to turn text into numbers:

- methods that consider the text to be an independent collection of words (“bag-of-words”) and
- methods that consider the order and syntactic structure of the text.

This last group of methods is often referred to as ‘embeddings’.

Under the bag-of-words assumption there are three ways in which one can transform text:

- Check if a word is present in a description and if so, record a 1 or otherwise a 0.
- Count the number of times a word is present in the text. This is called 'Term Frequency'
- Calculate the TF-IDF (Term Frequency – Inverse Document Frequency) scores.

This last way weighs the frequency with the inverse document frequency, the fraction of descriptions that contain the word. Words with the highest TF-IDF scores can be seen as the summary of the description, as these words are often present in the particular description, but not in all descriptions. These three methods can be applied to words, but also to character N-grams.

In the case of embeddings a d_{emb}-dimensional representation is made of every word in a text using its context words, the words that surround the particular word. The representation is a sequence of d_{emb} numbers that capture the semantics of the word. Words with similar context get similar representations. For example, the representations of 'walking' and 'running' are similar. There are many algorithms to generate embeddings; these vary in the way they consider context words. Popular ones are Word2Vec (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013), GloVe (Pennington, Sorcher, & Manning, 2014) and FastText (Joulin, Grave, Bojanowski, & Mikolov, 2016).

For our research we explore the possibilities of using embeddings from an external source, specifically the embeddings from (Tulkens, Emmery, & Daelemans, 2016). They trained embeddings on several datasets, among which Wikipedia, Sonar500 (Oostdijk, Reynaert, Hoste, & Schuurman, 2013) and a dataset with scraped data from Dutch *'.nl'* top domains. We use the Sonar500 embeddings as they strike a good balance between data size and breadth of embeddings. If the data size is large, it takes long to load the embeddings into memory. If the embeddings are mostly trained on one type of text, such as the Wikipedia set, they are not able to convey the right semantics.

When embeddings are used to classify sentences, the whole sentence is transformed. This can be done by concatenating the embeddings of all words in the sentence or by taking the unweighted average of the embeddings. A different option is to calculate a weighted average of the embeddings, e.g., by multiplying the TF-IDF document-term matrix by an embedding matrix. The first option results in a large amount of variables, considering an embedding per word is usually already 150 dimensional. In case of the second option a lot of information is lost, as only an aggregation of the original embeddings is kept. In the third option a TF-IDF document-term matrix with dimensions *n* by *k*, in our case *n* product descriptions with in total *k* words, is multiplied by an embedding matrix *k* by *d*, *k* embeddings with each *d* dimensions, to get to a document-embedding matrix with dimensions *n* by *d*. By taking this weighted average of the embeddings more emphasis is put on the more characteristic words of a description, determined by the TF-IDF score, as compared to an unweighted average.

A different way to use embeddings is by putting the complete sequence of embeddings for a k-word long sentence into a neural network and obtain a d_{enc}-dimensional representation of the sentence, where $d_{enc} < k \cdot d_{emb}$. We will discuss this further in Section 3.4.

2.2.4 Image data

Apart from the existing features, we tried adding new variables to obtain the right classification. In the case of clothing, it is often possible to gather the colour or size per product from the website. It turned out, however, while scraping the data, that these variables are not present for every vendor. This implies that for every vendor we would need a separate methodology, which is hard to fit into a production cycle. A variable that is present for every vendor is the picture of the product. That is why we chose to investigate this variable further.

For this research we scraped about 180.000 pictures of a clothes vendor by using the Python package Scrapy (Scrapy, 2019). Scrapy enables one to asynchronously make thousands of requests to websites, which speeds up webscraping. Approximately 8 hours were needed to collect all the images.

The images are transformed by means of a pre-trained neural network architecture called ResNet50 (He, Zhang, Ren, & Sun, 2015). The final representation of every picture is 2048-dimensional.

3. Methodology

After transforming the data, we use them in a machine learning model to classify products. Machine learning considers models that learn patterns from given examples. This is opposite to writing explicit instructions beforehand.

Roughly speaking, two types of machine learning models exist; unsupervised learning methods and supervised learning methods. In the case of supervised learning, one tries to find a model that describes the relation between input X, the features, and output y. A simple example is a model that can distinguish pictures of dogs from pictures of cats. Here the picture is the input and the label 'cat' or 'dog' the output. Using different combinations of input and output, the machine learning model finds patterns to distinguish whether the picture is a dog or cat.

In the case of unsupervised learning there are no output labels and there is just input X. The model tries to find groups or 'clusters' within this input. These clusters are often hard to interpret and not linkable to groups that are useful for the user of the model. Therefore, we only consider supervised learning for the CPI production.

Section 3.1 describes how a supervised learning model can be evaluated. Section 3.2 considers several ways to classify products. Sections 3.3 and 3.4 describe modern, 'deep learning' methods to classify products. Section 3.5 focuses on using unlabeled data to increase the accuracy of the models.

3.1 Evaluation

Before a supervised learning model can discover relations, it needs to be fed a large amount of hand labeled data. As manually labeling is a costly process, it is necessary to be as accurate as possible with the least amount of data. To evaluate correctness of a machine learning model, one splits the dataset in two: a dataset to train the model on, the training set, and a dataset to evaluate the model, a test set. By separating these sets, it is possible to test the model on a dataset the model has not seen before.

Apart from the correctness of the classifications of the model, it is also important that the model is feasible. A model that can classify all of het items correctly, but takes months to train is not useful in the CPI production process. The rest of this section is dedicated to methods to quantitatively express the aspects of correctness of the classification and feasibility.

The correctness of a model can simply be measured by comparing the predicted labels from the machine learning model to the actual labels. For this one uses the measures 'accuracy', 'precision', 'recall' and the 'F_1-score'. These can be explained using a confusion matrix as is shown in

Table 1. This is a confusion matrix for a binary classification; this means that the classifier can only tell if an item is part of a group (1) or not (0). The rows indicate the scenario in which an item is actually part of a group or not, and the columns indicate if the classifier predicts if the item is part of this group or not. In the cells the names of the four possible situations are shown.

Table 1: Confusion matrix for binary classification.

	Predicted label: 1	Predicted label: 0
Actual label: 1	True Positive (TP)	False Negative (FN)
Actual label: 0	False Positive (FP)	True Negative (TN)

In the case of accuracy one simply divides the number of correctly labeled products by the total amount of predictions, i.e.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FN + FP}.$$

This can lead to bias when the classes are not properly balanced. When one takes a classification model to recognize tumors on MRI scans, which is trained and tested on sets of images where 10% contains a tumor and 90% does not, the model can attain an accuracy of 90% by simply assigning the label 'No tumor' to everything. This is however not a well working model. In this case precision and recall are better ways to assess the quality of the model. These are defined as

$$\text{Precision}=\frac{TP}{TP + FP},$$
$$\text{Recall} = \frac{TP}{TP + FN}.$$

Precision looks at the ratio of the correctly predicted products in a group, compared to the total amount of products the model has placed in that category. Recall on the other hand takes the products that belong to a group and checks how many products are labeled as such. Therefore, precision focuses more on false positives, while recall focuses more on false negatives. To consider both, one can also take a weighted average. This is also called the F_b-score. It is calculated as

$$F_b = (1 + b^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(b^2 \cdot \text{Precision}) + \text{Recall}}.$$

The parameter *b* is usually taken to be equal to 1, after which we are left with the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

For this research we use accuracy, precision, recall and F_1-score to evaluate accuracy. As products are classified in more than two categories, we take the average of precision, recall and F_1 over all classes, weighted by the number of items in each class.

To measure feasibility, speed is most important; the algorithms need to be fast enough to be taken into a production cycle. To measure speed, we look at the elapsed real time during the computation of an algorithm.

3.2 Traditional models

In previous work implementations of the traditional models Logistic Regression, Naive Bayes, Random Forest and SVM have been used. We also use these models and in particular the implementations from the Python library Scikit-learn (Pedregosa, et al., 2011). As the models have often been applied in text classification, we leave the explanation of these models to reference works. Works we would recommend are (Bishop, 2006) and (Friedman, Hastie, & Tibshirani, 2001).

3.2.1 XGBoost

As an addition to the traditional models we look at the application of XGBoost (Chen & Guestrin, 2016). XGBoost is a machine learning system with many optimizations, not just in terms of accuracy, but also in terms of speed and scalability.

In its essence XGBoost is based on gradient tree boosting, where one iteratively trains regression or decision trees on the errors of the preceding models. As tree boosting methods are vulnerable to overfitting, XGBoost introduces both L_1 and L_2 regularization of tree nodes. Similarly to Random Forest, it also uses column sub-sampling, where only a subset of features are considered for each split. These additions to regular gradient tree boosting make XGBoost generalize well.

To increase speed, XGBoost contains a different splitting algorithm than regular tree boosting; where regular tree boosting goes through every possible splitting option and finds the one with the highest possible gain, XGBoost speeds up by choosing the best possible split from a smaller list of candidate splits. This is an approximation to the process used in regular tree boosting. The speed up as a consequence of this approximation is especially important when the explanatory variables are continuous. XGBoost's splitting also takes into account sparse data, achieving further speed gains by neglecting any zero-valued features.

Lastly, XGBoost keeps in mind the different computer architectures it runs on and the different data sizes it processes; it supports (i) parallelized learning, e.g. computations can be done on more than one CPU core, (ii) cache-aware access, causing fewer problems when gradient statistics do not fit into CPU cache, and (iii) efficient out-of-core computation when data needs to be fetched from one or multiple disks during computation.

3.3 Deep learning

To increase accuracy of classification, one can look at deep learning models. This group of models is comprised of many different ‘neural networks’. Neural networks emulate a large collection of neurons. This gives the models the possibility to detect more complex patterns. A disadvantage of these methods is that they use a large amount of internal parameters that have to be optimized during training of the model. This is taxing for the computer, both in terms of computational power and in terms of memory use.

In this research the neural networks are made and trained by means of the Python package PyTorch 1.0 (PyTorch, sd). This package allows the possibility to do calculations on the GPU if the GPU supports the NVIDIA technology CUDA (NVIDIA, sd) and the software is correctly configured.

The deep learning methods that are explored in this paper are, among others, feed-forward neural networks, recurrent neural networks (RNN), in particular the GRU architecture and convolutional neural networks (CNN).

3.3.1 (Feed-forward) Neural Networks

In a classifier based on a neural network, the data is fed through different layers of emulated neurons to calculate the probability that a product is in a certain category. In each layer the different attributes of the products are summed in a weighted way and the outcomes are fed to the next layer of neurons. In Figure 1 an example is shown of a perceptron inside a neural network. The attributes x_1 to x_n are combined in a weighted sum using the weights w_1 to w_n. This sum w is then processed by a non-linear activation function $f(x)$ to form the value a. This value can then be used as an attribute for a second layer of neurons. By choosing the number of layers and number of neurons per layer well, the model can discover patterns in the product characteristics through optimizing the weights.
A distinctive characteristic of the feed-forward neural network is that in every layer all attributes are taken into the weighted sum. Therefore it is also called a ‘fully-connected layer’ or ‘dense layer’.

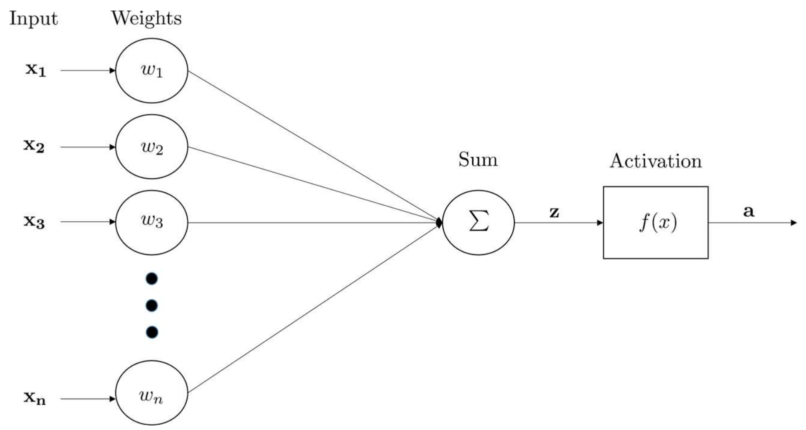

Figure 1: An example of a simple neural network in which weights are applied to input $\mathrm{x_i}$, summed and a non-linear activation function, $f(x)$, is applied.

3.3.2 Recurrent Neural Networks (RNN)

When classifying texts with sentence structures RNNs are often applied. By applying an RNN temporality is simulated; instead of classifying a text based on a collection of separate words, the output of the neuron corresponding to the first word is taken as input for the neuron of the second word. This is continued for the whole sequence. In this way the word order plays a bigger role in classifying the sentence.

Consider a RNN that is used to distinguish between positive sentences and negative sentences, i.e., binary sentiment analysis. One of the sentences can be 'not so good'. To use this as input we first need to turn the words of the sentence into their embedding representation, i.e., we replace [not, so, good] by [$\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$]. Then the RNN processes the sequence as in Figure 2.

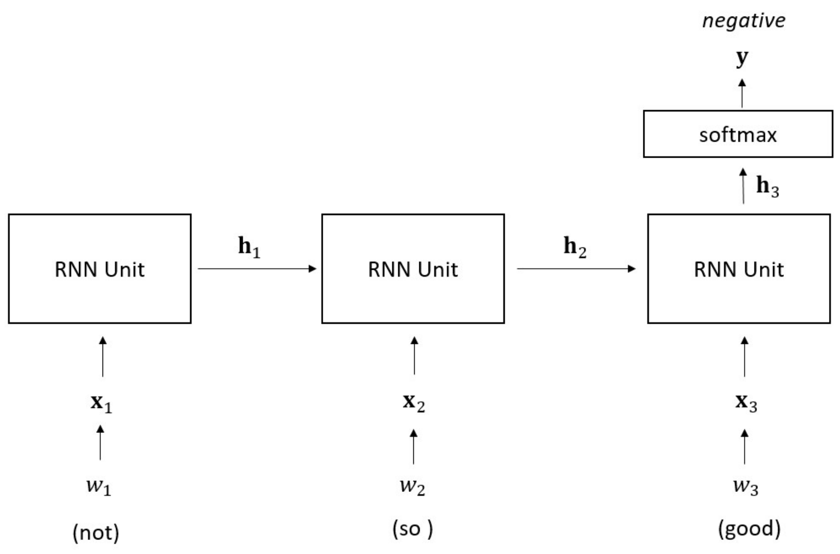

Figure 2: An example of a recurrent neural network.

In each step the embedding of a word is processed together with the state that is built on previous inputs. In this way, the model learns that when 'not' precedes 'good' the sentence can have a negative sentiment. Without the temporal aspect of the RNN the words would be evaluated separate from their word order. This could for example lead to this phrase being interpreted in the same way as the positive phrase 'good, not so busy'.

When training a neural network, our goal is to update the weights in order to reduce the classification error. This is done by taking the derivative of the error with respect to the weights. In order to calculate the necessary change in the weights at the start of the RNN, the chain rule is used through a process called 'backpropagation' (Bishop, 2006). This involves repeated multiplication of the gradients. With relatively deep networks, this can cause these products to vanish if the gradients are very small, making it impossible to train the initial layers of the network. To combat this, (Pascanu, et al., 2014) thought of a special type of recurrent unit called the Gated Recurrent Unit (GRU). In the rest of this paper we use this particular architecture when applying RNNs.

3.3.3 Attention Layers

A way to improve the accuracy of RNNs is by means of attention layers and specifically self-attention layers. Attention layer models are based on the principle that for classification the whole sequence is often not needed, but just some important details. To capture these details self-attention layers keep track of a context vector which determines whether a part of the input should be emphasized or not. This vector is trained together with the rest of the network.

3.3.4 Convolutional Neural Networks (CNN)

A different kind of popular neural network is the convolutional neural network (CNN). Convolutional neural networks use a set of filters to detect patterns in images or text. When applying them on text, the filter captures a small window of several, e.g. 3 or 5, words. This window is dragged over the text step by step and each time a weighted average is calculated of the words in the window, based on the weights in the filter. Through the incorporation of the context of a word, it is possible to capture semantics well and shared weights in the filters make it possible to recognize patterns, independently of where they occur in the text. A large advantage of CNNs over feed-forward neural networks is that the amount of parameters that have to be trained, is strongly reduced. Instead of assigning a weight to every word in a text and needing to optimize this weight, only the weights in the filter need to be trained.

Convolutional neural networks are most often applied on image data, but (Kim, 2014) shows they can also be used on text data, where they attain high accuracy. Besides that, they are known for their high computational speed. This should make them more appropriate for production cycles than different types of neural networks.

3.4 Combining encodings with other variables

In section 2.2.3 it is explained that with the use of a neural network, encodings can be generated using text data. When products are classified by means of a neural network, the generating of encodings can be integrated at many points in the process. In this research we try three. These combinations are described in the enumeration below and are shown in Figure 3, Figure 4 and Figure 5. To make it easier to talk about the different models, the models have been given names: 'Separate Encodings', 'Integrated Encodings' and 'Processed Encodings'. These three models all combine the encodings from the product description to the other variables, the brand and the ID, in a different manner.

In these images the dimensions are shown next to the representations, where
- *N* the number of products,
- S_{len} the length of a sentence,
- d_{emb} the dimensions of the word embeddings,
- d_{enc} the dimensions of the GRU encodings,
- n_{brands} the number of different brands,
- n_{grams} the number of unique N-grams within all IDs
- $n_{targets}$ the number of different classes within the dependent variable

The explored deep learning architectures are as follows:
- Separate encodings: We train encodings on the product description without the other variables by means of a GRU on the 'target', the dependent variable, in our case the COICOP classification. Subsequently, we concatenate the representation with the brand and the product ID and supply the combination to a different model, separately from the GRU that generated the encodings. This situation is shown in Figure 3.

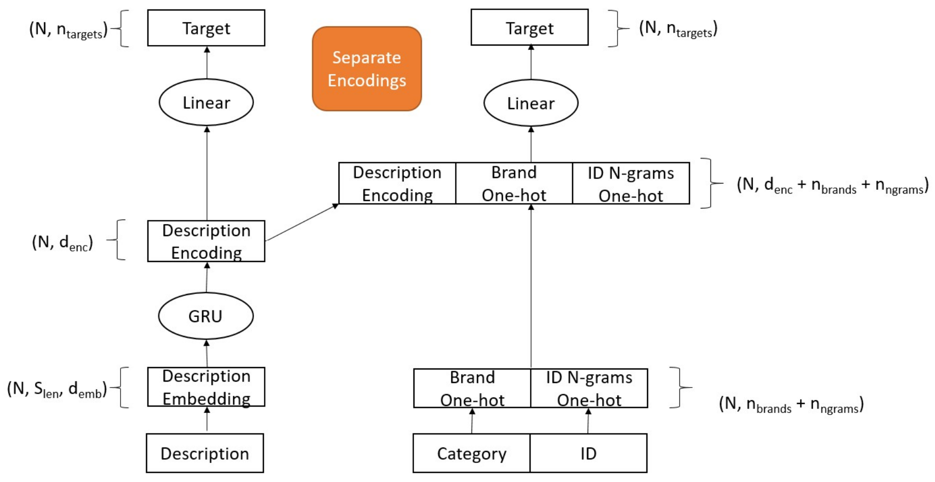

Figure 3: Separate encodings: The text encodings are separately trained by a GRU after which they are concatenated with features from the categorical variables and product IDs. This combination is then fed into a feed-forward network with 1 linear layer.

- Integrated encodings: In this model the generated encodings are concatenated with the other variables and fed into a linear layer. During training of the model the GRU is also trained; this means that during training the GRU is also adjusted and the encodings are aimed more at the target. The situation is shown in Figure 4.

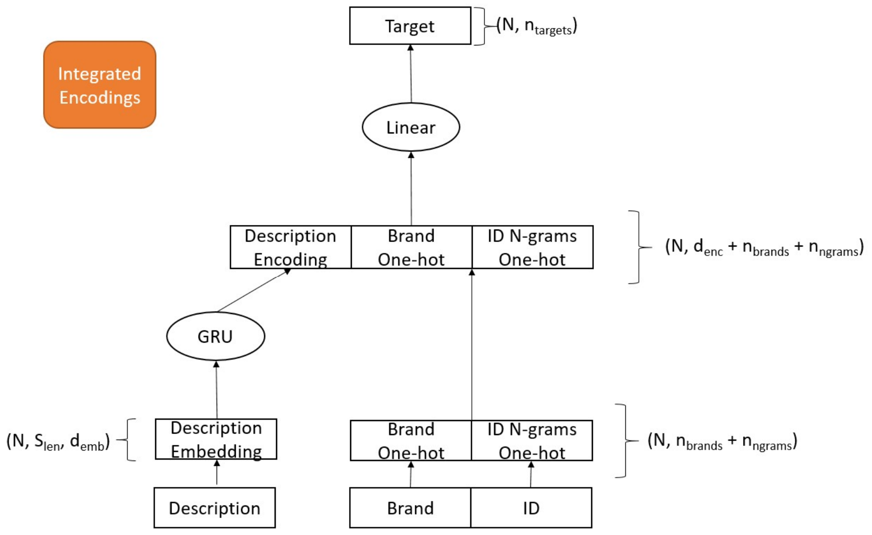

Figure 4: Integrated encodings: In this model the GRU is trained together with other parameters within the model. Within the model the GRU encodings are concatenated with other variables, after which the combination is put into a linear layer.

- Processed encodings: Just as in the integrated encodings model, this model also trains the GRU together with the rest of the neural network. However, in this model the word encodings and features of the categorical and ID variables are first processed by a linear layer and then concatenated, after which they are fed into a linear layer again. This situation is shown in Figure 55.

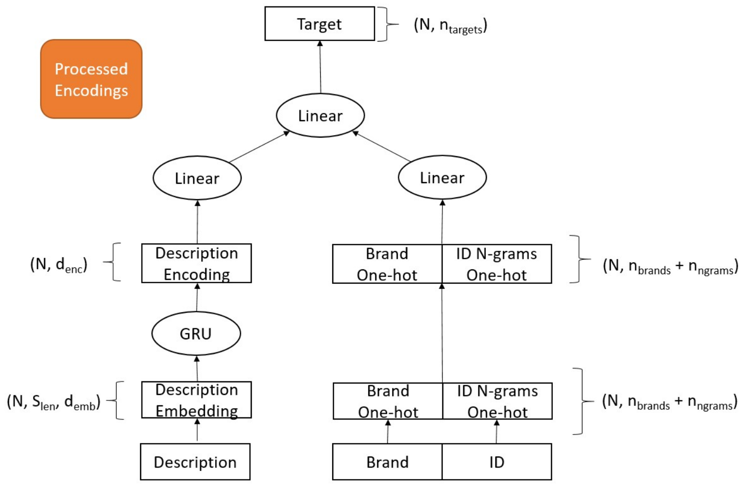

Figure 5: Processed encodings: In this model the GRU encodings are trained together with the other parameters within the model. Before the GRU encodings and the features from other variables are concatenated, they are first processed through linear layers.

3.5 Use of unlabeled data

In machine learning it is often true that the more training data you have, the better the model becomes. This wish for more training data is in practice not fullfilled due to the amount of time required to label data. In the previous sections, we described a number of models with which we try to get the best possible classifier using a limited amount of training data. On top of that, we look at methods to make use of unlabeled data to improve the classifier. Additionally, we investigate a way to choose unlabeled products for labelling that are likely to give the largest performance gain to the model. This last method will be referred to as incremental learning.

3.5.1 Incremental Learning

When using incremental learning we initially train a model on the labelled training dataset. By using this model we can calculate, for each product, the probability that a model assigns for the product to be in a specific category. Of these probabilities we take the maximum per product, noted as q_i, and average them over the whole dataset. The average value is referred to as the 'quality score' Q and is seen as a proxy for accuracy. If this value has not reached a certain threshold, then a weighted sample is taken from the unlabeled data and labelled. These new labeled points are added to the initial training set. The process is repeated until the threshold is reached.

3.5.2 Co-learning

A different way to use unlabeled data is by means of co-learning (Zhou & Goldman, 2004). In this case different models are used to determine which unlabeled products can be added to the labeled set with their predicted labels. The products for which a majority (or all) models give the same prediction, are added. In our case we experiment with different combinations of the models Naive Bayes, Random Forest, Logistic Regression, Linear SVM and XGBoost. A particular combination is trained on the train data, after which the combination determines which unlabelled points to add. Then, one of the models is trained on the final train data.

3.5.3 Tri-learning

Tri-learning (Zhou & Li, 2005) is a so called multi-view training method. In multi-view training methods different datasets with different features are used to obtain judgement about the unlabeled data. For tri-learning this is usually done by means of three bootstrap samples of the original data. A bootstrap sample is a random sample with replacement. For each of these bootstrap samples S_1, S_2 and S_3 a model is trained, which creates models m_1, m_2 and m_3. Subsequently, the models make predictions in pairs on the unlabeled data. When two models (e.g. m_1 and m_2) agree on a prediction, the sample is added to the bootstrap sample of the model that was not considered in the judgement (e.g. S_3). In this way it is prevented that a model takes its own prediction as training sample. The process is repeated until the models do not agree over any additional predictions.

The final models can be used on new data by means of majority voting, all models are applied on a new product and the prediction with the most votes is the final prediction. When none of the models agree, one can randomly choose one of the three predictions as the final prediction.

4. Results

This chapter contains the results of the models described in chapter 3.
To test the models, we use a dataset consisting of 320.000 items of clothing, to which we refer as dataset *K*. The data was collected by a web scraper. For each product it contains a product ID, brand and a short description of the product. All products are divided into a target demographic, Men, Women and Children, and each product is put into a category by the vendor, which are approximately at the level of the desired COICOP-categories. When each product is characterized by its target demographic combined with the vendor-specific category, 319 classes can be distinguished in dataset *K*.

In a realistic scenario only a small percentage of the complete dataset is labelled. Therefore, we check the performance of the machine learning and deep learning models on a small training set (10% of the data) and a large test set (90%). This split is done in a stratified way based on the target. For dataset *K*, this is a training set of about 32,000 products. For all experiments not concerning deep learning, the tests of the machine learning-models were repeated five times. For each of these five tests a new split in the train and test set was performed. The average of the evaluation measures is taken as the final score.

Most of the experiments were performed on a computer with an Intel Xeon E5-2650 v4 @ 2.20 GHz processor in the Statistics Netherlands' Centre for Big Data Statistics (CBDS) research environment. This CPU has 12 cores and 24 threads. In the rest of this paper the terms 'single core' and 'multicore' will be used to mean 'using one CPU thread' and 'using multiple CPU threads'. The computer has about 240 GBs of RAM. The computer does not contain a GPU, which is disadvantageous for the deep learning methods. These would profit the most from a GPU.

4.1 Scikit-learn models and XGBoost

For the scikit-learn tests we use scikit-learn implementations of Logistic Regression (LR), Naïve Bayes (NB), Random Forest (RF) and Support Vector Machines (SVM). Scikit-learn has two implementations of LR; a 'One-vs-Rest' implementation and a multinomial one. The former can be split over multiple CPU threads while the latter can only be trained on one thread.

In dataset *K* the brand of a product is treated as a categorical variable and turned into dummy variables. Short experiments showed that splitting the product ID in character N-grams of 2 characters works best. For the product description we also use character N-grams; a short experiment showed sets of 5 characters are ideal. In the case of N-grams for the product description, every product is described by about 52.000 features.

We evaluate the methods on accuracy, precision, recall and the F1-score. In addition, the total computation time is calculated. In this way we try to balance accuracy and feasibility within a production cycle.

Results for 1-core Scikit-learn implementations are in Table 2 while results for multicore (n_{threa} =20) implementations are in Table 3.

Table 2: Results of scikit-learn implementations of machine learning algorithms on dataset K. For computation time the format hh:mm:ss is used. The number of threads is equal to 1.

Model	Accuracy [%]	Precision [%]	Recall [%]	F_1-score [%]	Computation time
LR - Mult.	91.94	91.97	91.94	91.70	00:10:05
NB	66.07	69.49	66.07	59.81	00:00:26
RF	88.27	88.49	88.27	87.93	00:00:51
LinearSVC	91.81	91.77	91.81	91.68	00:00:46

Table 3: Results of scikit-learn implementations of machine learning algorithms on dataset K. For computation time the format hh:mm:ss is used. The number of threads is equal to 20.

Model	Accuracy [%]	Precision [%]	Recall [%]	F1-score [%]	Computation time
LR - OvR	92.14	92.27	92.14	91.91	00:00:43
RF	88.36	88.62	88.36	88.01	00:00:28

The One-vs-Rest implementation of Logistic Regression achieves the highest accuracy, precision, recall and F_1-score. The F_1-score is 91.91%. Random Forest and LinearSVC are not much worse with F_1-scores of 88.01% and 91.68%. This is in contrast with the Naive Bayes approach, that only achieves an F_1-score of around 60%.

Considering speed of the algorithms, we see the opposite result, where Naive Bayes is fastest, shortly followed by the multicore implementation of Random Forest. LR-OvR, LinearSVC and single-core RF are about 20 to 25 seconds slower than NB and RF. LR-Mult. is significantly slower than all the aforementioned.

Based on these results, we would recommend LR-OvR when it is sufficiently fast for a production cycle. When the amount of records is larger than the 320.000 in dataset K, Random Forest can be a solution because of the balance between speed and accuracy.

4.1.1 XGBoost

In addition to the scikit-learn models we applied the gradient tree boosting algorithm XGBoost to dataset K. Gradient tree boosting involves repeated training on residuals of previous decision or regression trees. The result of the tests of XGBoost can be seen in Table 4.

Table 4: Results of XGBoost on dataset K. For computation time the format hh:mm:ss is used.

Model	Accuracy [%]	Precision [%]	Recall [%]	F1-score [%]	Computation time
XGBoost	91.77	92.23	91.77	91.60	00:14:54

In terms of accuracy, XGBoost is on par with the best performing scikit-learn models, but reaches this accuracy in nearly 15 minutes rather than 45 seconds. The XGBoost algorithm has large number of hyperparameters which can be varied in order to possibly achieve a better performing classifier for a certain dataset. In practice, Logistic Regression may be a safer option, as it has fewer hyperparameters to tune and is therefore more easily generalized to other vendors.

4.1.2 Using external embeddings

When using the character N-grams text transformation on the product descriptions, information about the meaning of words gets lost. To investigate what influence retention of the semantic information has on the performance of machine learning models, we made a document-embedding matrix for the product description using word embeddings from an external source and a TF-IDF transformation of the product description. For the embeddings we used the 160-dimensional SoNaR-500 embeddings (Tulkens, Emmery, & Daelemans, 2016). Together, the N-gram transformations and the document embedding matrix form the set of features for the description. The brand and product ID are still processed by means of the character N-grams transformations of the product descriptions. All features are fed into a scikit-learn Logistic Regression implementation. The results are shown in Table 5.

Table 5: Results of LR - OvR with external embeddings. For computation time the format hh:mm:ss is used.

Model	Accuracy [%]	Precision [%]	Recall [%]	F1-score [%]	Computation time
Ext. embeddings	60.97	58.20	60.83	58.19	00:05:15
Ext. emb. + Char. N-grams	92.09	92.21	92.09	91.86	00:05:46

Table 5 shows that using external embeddings does not lead to an improvement in accuracy of the model. It can be that adding the document-embedding matrix does not provide sufficient new information to the model to achieve an increase in accuracy.

The computation time does increase from 45 seconds to 5 minutes due to the need to load the external embeddings from a file. Adding the weighted average of the embeddings does therefore not have a positive influence on the total performance of the Logistic Regression model on dataset *K*.

4.2 Deep learning: PyTorch

Table 6 shows the results of deep learning tests on dataset *K,* performed using the PyTorch framework in the programming language Python. Just as before, for all these tests the brand and the product ID were transformed by respectively adding dummy variables and applying character N-grams transformations. The transformation of the product description differs depending on the model used.

The first row of Table 6 shows the results of the feed-forward neural network (FF-NN); when using this model the product description is transformed by means of a character N-grams transformation of size 5, as done for the scikit-learn models. For the other deep learning models the product descriptions are processed through self-training embeddings and word encodings by means of a unidirectional GRU. The word embeddings are chosen to be 150-dimensional; for the word encodings 128 dimensions were chosen.

Note that for recurrent neural networks the input has to be of a constant size. To adhere to this assumption, sentences can be either padded with zeros if they are too short or cut if they are too long. In this paper the input length was set equal to 10 words.

Table 6: Results of PyTorch implementations of several neural networks on dataset K. For computation time the format hh:mm:ss is used.

Model	Accuracy [%]	Precision [%]	Recall [%]	F_1-score [%]	Computation time
FF-NN	91	91.59	91.50	91.32	09:22:53
Separate enc.	91	91.72	91.00	90.93	01:35:15
Integrated enc.	90	90.82	90.73	90.47	01:58:44
Processed enc.	91	91.34	91.36	91.16	02:07:50
Attention	91	91.77	91.62	91.51	01:53:07
CNN	90	91.00	90.96	90.76	01:47:45

In the case of the deep learning models, the best performing models in terms of the F_1-score are the model with attention layers and the feed-forward neural network with an F_1-score of respectively 91.51% and 91.32%. The feed-forward neural network however is on the extreme end of the spectrum in computation time with time of approximately 9 1/2 hours. The fastest model, with a computation time of around 1 1/2 hours, is the one where encodings are trained separately from the rest of the model.

The accuracies of deep learning models are comparable to those of Logistic Regression, Random Forest and SVM, but the deep learning approaches are slower. Other informal experiments with GPU hardware showed a smaller difference between these two groups of models in terms of speed. We would therefore recommend replicating these experiments in a setup with more GPU power.

4.2.1 Adding image data

It is also possible to add information from pictures of products into the classification algorithms. To test the efficacy of this approach we scraped the product information and the images of 180.740 products from the website of the same vendor as for dataset K and processed their images to a 2048-dimensional representation using a pretrained ResNet50-network. The representation is subsequently concatenated with features from one hot encoding the brand and performing character N-grams transformations on the product ID and product description as before. The complete set of features can then be processed by a machine learning model. For one test we used a feed forward neural network to classify the products and for a different test we used the LinearSVC implementation of Support Vector Machines. As we used a dataset different from dataset K, we also did a reference test on the same dataset, without the features from the image data.

To process the image data, more computational power was needed than available in the CBDS infrastructure. Therefore, these experiments were performed on different hardware, namely a computer with an Intel Core i7-8750H and an NVIDIA GTX 1050Ti GPU. For the neural network, comparable to the Feed Forward Neural Network from Table 6, optimal performance was obtained after 5 epochs.

Table 7: Results of PyTorch implementations of several neural networks on dataset K with image data. For computation time the format hh:mm:ss is used.

Model	Accuracy [%]	Precision [%]	Recall [%]	F_1-score [%]	Computation time
LinearSVC (without image data)	91.48	91.46	91.48	91.33	0:00:18
LinearSVC (with image data)	92.36	92.14	92.36	92.10	0:32:46
Neural network (with image data)	91.36	90.69	91.36	91.02	2:32:35

Best performance was achieved by the LinearSVC with image data. Its F_1-score is approximately one percent point higher than that of the LinearSVC without image data. The neural network with image data performed worse than both the LinearSVC with and without image data.

4.3 Effectivity of deep learning

Aforementioned experiments show that traditional methods outperform neural network approaches on dataset *K*. This does not agree with literature on text classification, that shows neural networks outperform traditional methods on benchmark datasets (Yang, et al., 2016; Kim, 2014). It begs the question whether inferior performance of neural networks can be attributed to the difference in datasets or the difference in implementation. We hypothesize that it is due to the former, i.e., neural networks do not perform well when the texts are short and non-cohesive as these networks mostly depend on semantics.

To investigate this hypothesis, we test our deep learning implementations on datasets with longer descriptions, namely several English open source datasets. These datasets only contain text and no other variables. Unfortunately these types of datasets were not avaiable in Dutch.

The first dataset, dataset *Y*, contains approximately 1.46 million question-answer pairs with corresponding categories. For the classification the questions and answers were concatenated to form one text with the category as target. The dataset contains 10 categories in total, namely 'Society and Culture', 'Science and Mathematics', 'Health Care', 'Education', 'Computers and Internet', 'Sport', 'Business and Finance', 'Entertainment and Music', 'Family and Relationships' and 'Politics and Government'. To limit the computation time, we took a random sample of 10% and split it into 90% train data (131,400 pairs) and 10% test data (14,600 pairs).

The second dataset, dataset *Z*, contains 50,000 reviews of films with the corresponding sentiment (positive or negative). 25,000 of these reviews are labeled as positive and 25,000 as negative. Again the dataset is split into 90% train data (45,000 reviews) and 10% test data (5,000 reviews).

As the datasets merely contain text, only one type of deep learning architecture was used; where the model is first applied followed by respectively a linear layer with 128 neurons and a softmax layer.

Experiments showed the CNN attained the best performance with 4 epochs, while the RNN and Attention Layer model worked well with 2 epochs. Table 8 and Table 9 show the results of the deep learning models on datasets *Y* and *Z*.

Table 8: Results of implementations of several machine learning algorithms on dataset *Y*.

Model	Accuracy [%]	Precision [%]	Recall [%]	F1-score [%]
LR - OvR	68.43	68.14	68.43	68.26
RNN	68.97	68.64	68.97	68.80
Attention	69.64	69.68	69.64	69.66
CNN	69.53	69.12	69.53	69.32

Table 9: Results of implementations of several machine learning algorithms on dataset ***Z***.

Model	Accuracy [%]	Precision [%]	Recall [%]	F1-score [%]
LR - OvR	90.68	90.68	90.68	90.68
RNN	91.24	91.25	91.24	91.24
Attention	91.22	91.25	91.22	91.23
CNN	91.10	91.10	91.10	91.10

All deep learning models perform better than the traditional approach of applying Logistic Regression on character N-grams transformed descriptions. The Attention Layer model achieves the best performance on dataset Y while RNN and Attention share the first spot on dataset Z. These results support our hypothesis that the bad performance of deep learning models on dataset K can be attributed to small non-cohesive descriptions.

4.4 Unlabeled data

We described three algorithms to use unlabeled data for more accurate classification: incremental learning, co-learning and tri-learning. In this section we show the results of these algorithms on dataset K.

In each section a three-way split is made in a training set, hold-out set and test set. The training set is 10% of the total data set (about 32.000 records). It is used to train the initial model. Afterwards, the models are applied on the hold-out set, 87.5% of the data, to obtain more training points and training is repeated for the new training set. For incremental learning and tri-learning, this is done iteratively till a stable point is reached. For co-learning we only add new samples once.

Lastly, the final model or combination of models is used to classify on the test set, 2.5% of the data. The evaluation measures accuracy and F_1-score are reported based on these classifications. As the test set has a different size from previous experiments, we also report results for reference models.

4.4.1 Incremental learning

The incremental learning method was tested using the LR-OvR model. After initially being trained on the train data set, the quality score Q of the classification of the unlabeled data is determined. If the threshold of 0.92 is not reached, 1000 products are drawn from the unlabeled dataset through a weighted random sample, and added to the train data set along with their label. The weights are given by $w_i = \frac{1}{q_i^\alpha}$ for the i^{th} product. After the incremental learning method has reached the threshold value, the accuracy and the F_1-score are calculated on the test dataset and the size of the train dataset is determined. Subsequently a train dataset and test dataset of the same sizes are randomly drawn from the total dataset and again an LR-OvR model is trained and tested. This is called the 'direct' method. The tests of the incremental learning method and the direct method are repeated five times for the values α = 4 and α = 8 and the averaged results are shown in Table 10. This table tells us that for both tests the incremental learning method increases the accuracy and the F_1-score by more than a percentage point. It is also shown that the tests with α = 8 need 4000 fewer labeled examples than the tests with α = 4.

Table 10: Resuls of the tests of the incremental method using the LR-OvR model on dataset K. For computation time the format hh:mm:ss is used.

α	# Iterations	Direct: Accuracy [%]	Direct: F_1-score [%]	Incremt.: Accuracy [%]	Incremt.: F_1-score [%]	Computation time
4	15	93.92	93.77	94.31	94.19	00:10:52
8	11	94.09	93.94	94.73	94.59	00:08:13

4.4.2 Co-learning

Co-learning uses one type of main model and three types of supporting model. First, the main model and the three supporting models are trained on the training set. Then all of the models are applied on the hold-out set to get to four predictions for each data point. The points on which all the models agree are added to the training set to get to a larger training set. At the end, the main model is trained on this final training set and applied on the test set.

Co-learning is done with LR - OvR, LinearSVC and XGBoost as main models, as these have shown the best performance in previous experiments. The supporting models are chosen from Naive Bayes, Random Forest, LR-OvR, LinearSVC and XGBoost. We include low performing models, such as Naive Bayes, into the supporting models, as we hypothesize an ensemble of models might benefit from weak learners.

The scores are reported in Table 11.

Table 11: Results of co-learning on dataset K. For computation time the format hh:mm:ss is used. In the 'Supporting models' column the abbreviations LinSVC, XGB and LR are used for LinearSVC, XGBoost and LR - OvR respectively. For the experiment with the XGBoost model with an asterisk only the four best performing tests are used for the averaged result.

Model	Supporting models	Main: Accuracy [%]	Main: F_1-score [%]	Co: Accuracy [%]	Co: F_1-score [%]	Computation time
LR - OvR	NB, RF, LinSVC	92.26	91.97	92.88	92.63	00:02:51
LR - OvR	NB, RF, XGB	92.19	91.95	92.69	92.44	00:16:47
LR - OvR	RF, LinSVC, XGB	92.10	91.78	93.59	93.42	00:17:53

LinearSVC	LR, NB, RF	91.97	91.76	92.99	92.85	00:06:39
LinearSVC	LR, RF, XGB	91.88	91.68	93.71	93.62	00:23:45
LinearSVC	NB, RF, XGB	91.83	91.65	92.67	92.55	00:20:14
XGBoost	LR, NB, RF	91.99	91.77	84.10	83.39	00:46:15
XGBoost*	LR, NB, RF	92.06	91.83	91.61	91.31	00:49:21
XGBoost	LR, RF, LinSVC	91.79	92.40	93.13	91.56	00:59:46
XGBoost	NB, RF, LinSVC	92.01	91.73	91.86	91.52	00:49:48

The most accurate combination is using LinearSVC as main model and LR, RF and XGB as supporting models. The combination achieves an F_1-score of 93.62%. The fastest combination is LR-OvR as main model and NB, RF and SVC as supporting models. This combination only takes 2 minutes and still achieves better performance than any single model, namely an F_1-score of 92.63%.

In general, including Naive Bayes leads to worse accuracy than not including it, as the three best performing combinations all do not contain Naive Bayes. Therefore, we recommend against it.

During the experiment with XGBoost as the main model and LR-OvR, Naive Bayes and LinearSVC as co-models, one of the five test runs used to derive the average result showed drastically lower scores when the co-models were applied. In this run the F_1-score for the main model was 91.56% while it was 51.74% after the co-learning procedure was applied. The assumption is that this lower score resulted from an unfortunate train-test split. Due to the time constraints on this project a definite origin of this anomaly was not identified. The averaged result of the other four runs are shown, denoted by an asterisk, in Table 11.

4.4.3 Tri-learning

Tri-learning uses bootstrap samples to get to different views of the data. On each view a model is trained after which pairs of models determine whether a data point should be added to the training set of the third model. This process is repeated until no more new training points are added to one of the models.

We try tri-learning with LR - OvR and with LinearSVC as they were the best performing models in terms of accuracy. We also try Random Forest, but limit the number of trees to one per model. Using one tree increases the variance between the models and therefore limits the number of observations the models agree on. When simple random trees agree, we should be more sure about the particular classification.

The results of tri-learning are reported in Table 12.

Table 12: Results of tri-learning on dataset K. For computation time the format hh:mm:ss is used. The simple random forest uses only one decision tree.

Model	Ref: Accuracy [%]	Ref: F_1-score [%]	Tri: Accuracy [%]	Tri: F_1-score [%]	Computation time
LR - OvR	92.49	92.17	92.28	91.83	01:32:51
LinearSVC	91.39	91.23	91.37	91.06	05:21:06
Random Forest - Simple	71.42	71.25	81.35	79.97	00:22:43

Both LR-OvR and LinearSVC achieve similar performance to the benchmark after tri-learning, in around 1 1/2 hours and 5 1/2 hours respectively. In the case of the Random Forest with one tree, there is a considerable improvement in terms of accuracy, about 10 percent points. The accuracy is, however, still worse than LR - OvR and LinearSVC without tri-training. Based on these results, we do not recommend tri-learning for product classification.

5. Conclusion

We have discussed several ways to classify products into product categories by using descriptions, IDs, categories and image data. The models that were used ranged from traditional machine learning models to deep learning approaches. We also experimented with the use of available unlabeled data to improve accuracy.

It turned out that for the data transformations, when image data was not considered, simply one-hot encoding the categorical data and using frequencies of character N-grams performed best. Using this transformed data, the best performing model was Logistic Regression (One-vs-Rest), which delivered its classification in under a minute of computation time. It could be possible that by tuning the hyperparameters of the XGBoost model a better performing model can be produced. This experiment was outside of the scope of this research project. The use of XGBoost over Logistic Regression would however come at the cost of a longer computation time. The disappointing performance of deep learning models can be attributed to the short length and lack of cohesion in the descriptions, as analysis on other datasets showed that our implementations of deep learning models did perform best when sentences were longer.

To include image data, deep learning needs to be considered to reliably extract features from images. Our experiments showed that using image features from a pre-trained neural network already increased performance by a significant margin. Experimenting with different pre-trained neural networks or adjusting parameters of the pre-trained network might lead to further performance improvements. We did not, however, have the resources available to perform such experiments.

Unlabeled data can improve performance of classifiers, both through co-learning, where an ensemble of models is used to determine which unlabeled observations to add and through incremental learning, where one determines the next batch of observations to label by means of a quality metric based on the probability the model assigns to a certain prediction. We recommend using both approaches.

Most of the work in this paper was dedicated to dataset K; a dataset of clothing from one particular vendor. The results on this dataset do not tell whether models can generalize to other clothing vendors or different kinds of vendors, e.g., supermarkets. We suggest mixing datasets and trying to achieve more general models. Besides that, we did not consider datasets at different points in time. As vendors constantly change their set of products, when and how to update a model should be investigated.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473.*

Barb, F. D. (2018). *Testing supervised probabilistic machine learning classification algorithms for the automatic mapping of GTINs to ECOICOPs.* EUROSTAT.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.*

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International conference on knowledge discovery en data mining*, 785-794.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The elements of statistical learning.*

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recoginition. *arXiv preprint arXiv:1512.03385*.

Howard, J. (sd). *FastAI v1*. Opgeroepen op Januari 21, 2019, van FastAI: https://www.fast.ai/

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of trics for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., . . . Liu, T. Y. (2017). LigtGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, 3146-3154.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 1097-1105.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 3111-3119.

NVIDIA. (sd). *CUDA Zone*. (NVIDIA) Opgeroepen op Februari 26, 2019, van https://developer.nvidia.com/cuda-zone

Oostdijk, N., Reynaert, M., Hoste, V., & Schuurman, I. (2013). The construction of a 500-million-word reference corpus of contemporary written Dutch. In *Essential speetch and language technology for Dutch* (pp. 219-247). Berlin: Springer.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *International conference on machine learning*, 1310-1318.

Pascanu, R., Mikolov, T., Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., . . . Lerer, A. (2017). Automatic differentiation in PyTorch. *31st Conference on Neural Information Processing Systems (NIPS).* Long Beach, CA. USA.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research, 12(Oct)*, 2825-2830.

Pennington, J., Sorcher, R., & Manning, C. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532-1543.

PyTorch. (sd). *PyTorch*. (PyTorch) Opgeroepen op Februari 26, 2019, van https://pytorch.org/

Ruder, S., & Plank, B. (2018). Strong baselines for neural semi-supervised learning under domain shift. *arXiv preprint arXiv:1804.09530*.

Scrapy. (2019, januari 30). *Scrapy 1.6.0 - Release notes*. (Scrapy) Opgeroepen op maart 21, 2019, van https://docs.scrapy.org/en/latest/news.html

Singh, D., & Reddy, C. K. (2015). A survey on platforms for big data analytics. *Journal of big data 2(1)*, 8.

Tulkens, S., Emmery, C., & Daelemans, W. (2016). Evaluating unsupervised Dutch word embeddings as a linguistic resource. *arXiv preprint arXiv:1607.00225*.

van den Heuvel, E. G. (2018). *Classification of goods for CPI using machine learning techniques.* Den Haag: CBS.

Walschots, J. (2019). *Het mandje van de consumentenprijsindex, de bestedingen van consumenten en de meting van de CPI in 2019.* Den Haag: Centraal Bureau voor de Statistiek.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical attention networks for document classification. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1480-1489.

Zhou, Y., & Goldman, S. (2004). Democratic co-learning. *16th IEEE International Conference on Tools with Artificial Intelligence*, 594-602.

Zhou, Z. H., & Li, M. (2005). Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge & Data Engineering*(11), 1529-1541.

About discussion papers

Colophon

Publisher
Centraal Bureau voor de Statistiek
Henri Faasdreef 312, 2492 JP Den Haag
www.cbs.nl

Prepress
Statistics Netherlands, CCN Creation and visualisation

Design
Edenspiekermann

Information
Telephone +31 88 570 70 70, fax +31 70 337 59 94
Via contactform: www.cbs.nl/information