mathematics

MDPI

Article

Automatic Product Classification Using Supervised Machine Learning Algorithms in Price Statistics

Bogdan Oancea

Department of Applied Economics and Quantitative Analysis, University of Bucharest, 030018 Bucharest, Romania; bogdan.oancea@faa.unibuc.ro

Abstract: Modern approaches to computing consumer price indices include the use of various data sources, such as web-scraped data or scanner data, which are very large in volume and need special processing techniques. In this paper, we address one of the main problems in the consumer price index calculation, namely the product classification, which cannot be performed manually when using large data sources. Therefore, we conducted an experiment on automatic product classification according to an international classification scheme. We combined 9 different word-embedding techniques with 13 classification methods with the aim of identifying the best combination in terms of the quality of the resultant classification. Because the dataset used in this experiment was significantly imbalanced, we compared these methods not only using the accuracy, F1-score, and AUC, but also using a weighted F1-score that better reflected the overall classification quality. Our experiment showed that logistic regression, support vector machines, and random forests, combined with the FastText skip-gram embedding technique provided the best classification results, with superior values in performance metrics, as compared to other similar studies. An execution time analysis showed that, among the three mentioned methods, logistic regression was the fastest while the random forest recorded a longer execution time. We also provided per-class performance metrics and formulated an error analysis that enabled us to identify methods that could be excluded from the range of choices because they provided less reliable classifications for our purposes.

Keywords: automatic product classification; price statistics; FastText skip-gram; logistic regression; support vector machines; imbalanced multi-class classification

MSC: 68T01

Citation: Oancea, B. Automatic Product Classification Using Supervised Machine Learning Algorithms in Price Statistics. *Mathematics* **2023**, *11*, 1588. https://doi.org/10.3390/math11071588

Academic Editors: Zhao Kang, Ioannis G. Tsoulos and Ivan Lorencin

Received: 16 January 2023
Revised: 1 March 2023
Accepted: 22 March 2023
Published: 24 March 2023

1. Introduction

With the advent of the technological revolution, big data has been targeted as having immense potential for obtaining more time-related and relevant statistics at a lower cost. One of the areas where big data has been adopted is for the computation of consumer price index (CPIs). Several authors ([1–7]) have reported the potential to integrate new data sources such as web-scraped data and scanner data into the computation of CPIs in order to augment the traditional data used for calculation. The main advantages of using such data sources for CPI calculations consist of increasing its timeliness and relevance while reducing the costs of the data collection, objectives that are in agreement with the requirements for the modernization of official statistics.

Following this trend, in [8], we described a set of tools we developed to collect data from major national e-commerce sites, and since their development, we have collected around 50,000 records on a weekly basis, thus building a very large dataset.

Since CPIs are computed as a weighted average of prices for a basket of goods and services that are representative of aggregated consumer spending, the first step after data collection is to group the products according to the classes of goods and services that make up the basket. In a classical approach, when the number of products from each

group (or category) is limited, products are labeled manually by human experts, but when using big data sources, manually labeling these records is impossible due to the high volume of data; therefore, an automatic classification process should be used. In this paper, we describe a process of automatic product classification in a multi-class setting, using a series of machine-learning techniques, in order to transform the price data for CPI computation. The information collected from e-commerce sites included the name and a textual description of each product, together with the corresponding price. These records were grouped according to the product classes that formed the basket, and we first transformed the product names into numeric vectors and then applied an automatic classification method to classify the vectorized names. As a result, we could select both the vectorization method that produced the best separation between product classes and the classification algorithm with the best performance. Once the products were grouped, the computation of the CPI could be conducted for each product group separately and then aggregated, based on the weights of each group.

The process of product classification has been streamlined, reducing it to a text document classification problem, which is a hotspot in the research. Features such as the text length and the purpose of the text can significantly influence the performance of a classifier. For example, classifying a set of newspaper articles can be very different from classifying a set of products by their description. Therefore, there is no *standard* method for performing such a classification, and the decisions are determined on a per-case basis.

While automatic classification using machine learning methods has been reported by several authors in different fields, ranging from sentiment analysis (see for example [9] and the references therein) to scientific literature classification [10], medicine ([11], object recognition [12], and diverse price indices computation [13], in this paper, we describe our approach for classifying a set of products by their names according to an international classification used for CPI computation. We began by collecting a relatively small sample of records (2853 products) from the price database that we built and manually labeled each product with its corresponding class, using the European Classification of Individual Consumption according to Purpose (ECOICOP) international product classification [14], with five-digit classes. A description of the datasets is presented in Section 3. Machine learning methods handle numerical data, but our product names were text data; therefore, before using any machine learning methods to classify the products, we needed to transform text data into numerical data. For this task, we used a series of techniques called *word embedding*. Therefore, we built several numerical vector representations for each product name in our dataset, each representing an embedding technique. The concept that we followed was to be able to choose not only the best machine-learning-classification method but also the word-embedding technique that was best suited for our needs (i.e., produced the best differentiation between product classes). In an exploratory data analysis, we built 2D visualizations for each set of numerical vectors, corresponding to the product names, to determine which embedding technique produced the best separation between the classes. Then, we proceeded to apply a set of machine-learning-classification methods and computed the performance metrics for each. All the embedding techniques and classification methods used in this experiment are presented in Section 4, while in Section 5, we present the results, and in Section 6, we discussed compared our results with those of other similar studies. The performance of a classification model greatly depended on the value of its parameters. Being a pilot study, we wanted to ensure the operational time was under acceptable limits. Therefore, we used a grid search to select the optimum values of the hyper-parameters for only a few selected methods, and the operational time was approximately 24 h. This paper contains a section dedicated to an error analysis and ends with the final conclusions and directions for future work.

2. Related Work

Following recent technological developments, official statistics bureaus, which are typically in charge of the CPI computation in every country, have adopted machine learning

methods for product classification. while these methods are still in the experimental phase, there have been some notable results presented in this area. Therefore, Roberson reported in [15,16] the results of a study regarding the product classification using The North American Product Classification System based on a description of each product, showing that the automatic classification procedure achieved an accuracy over 90%. Martindale et al. described in [17] the process of using web-scraped data records regarding clothes for CPI computations, using the COICOP5 classification. The authors started by manually labeling a small subset of products to build a training dataset, then enlarged this dataset using fuzzy matching techniques, based on the Levenshtein distance, partial ratio, and the Jaccard distance. They also used label propagation and spreading techniques that were semi-supervised to label the products. Having a large labeled set of products, three machine learning methods were used to build an automatic classifier, namely support vector machines with a non-linear kernel, decision trees, and random forest. The results showed good performance, with a precision between 0.86 and 0.90 and an F1-score between 0.80 and 0.87, depending on the classification method and the word-embedding technique used. The authors concluded their work with a discussion on the performance metrics of the classifiers, stressing the impact of the incorrectly excluded and included products on the price index.

Another study on product classification for price statistics was described in [18]. Here, the author showed how different datasets from several sources were combined in a training dataset that could later be used for classification models. Only two classification algorithms were used, random forest and logistic regression, building the word embedding with count vectorization and term-frequency–inverse-document-frequency methods. On the test set, the best precision was obtained with random forest (0.87), and this method also had a better F1-score than logistic regression (0.86 versus 0.81).

Myklatun [19] presented the results of another study developed at Statistics Norway, where data for food and non-alcoholic beverages were automatically classified using a regularized logistic regression model, a naïve Bayes classifier, and a support-vector-machine model. The best accuracy of the classification was obtained with support vector machines at 90.2%, followed by regularized logistic regression at 89.3% and naïve Bayes at 87%. The author reported that using the automatic classification significantly reduced the time consumed by the CPI calculation.

Automatic product classification has also been used in commercial applications, as presented in [20], where the authors described a process of using the naïve Bayes method to classify two sets of products presented on a commercial website. The authors described their vectorization method that used the bag-of-words technique and analyzed how different pre-processing techniques, such as stemming, stop-word removal, number removal, etc., influenced the accuracy of the predictions. The authors reported an accuracy of 79.6% for naïve Bayes on one of the two datasets involved in their study. They also experimented with kNN and a tree classifier that provided an accuracy of 69.5% and 86%, respectively, but they argued that the trade-off between the accuracy and the operational time indicated naïve Bayes was a better method for their purposes.

Other works [21–23] also discussed the automatic classification of text data, and several authors emphasized that when using such methods, the costs of data processing and the time required for this task were reduced.

However, the current studies addressing with the problem of product classification for CPI computations have limitations:

- Most considered only simple embedding techniques, such as count vectorization or term frequency–inverse document frequency, which have a significant drawback: they cannot be used with words not in a standard dictionary, so that when we presented the classifier with a new product not used in the training set, the embedding process had to be repeated. The only study that extended the vectorization techniques to a method capable of handling with words not in a standard dictionary was [17].

- The number of machine-learning-classification methods used in these studies was rather limited. Most existing studies were limited to only two or three classification methods with logistic regression, random forest, and support vector machines being widely used, though a few authors reported that they used methods such as naïve Bayes or kNN.
- The metrics used to compare the classification performances of different methods were limited to the classical accuracy and F1-scores, even when the datasets were imbalanced, which requires special attention to the classification results.
- An error analysis was also absent in most of the existing studies.

Based on these limitations, we attempted to extend and improve these previous studies by:

- Using a wide range of the existing embedding techniques: Count vectorization; term frequency–inverse document frequency; Word2Vec (both CBOW and skip-gram) with two variants for computing the vectorization of a product name; FastText with both CBOW and skip-gram variants; and GloVe, a method that was not tested at all in the previous studies.
- Using a wide range of classification methods: We used a total of 13 methods, including 7 variants of decision-tree-based methods, neural networks, support vector machines with different kernels, multinomial naïve Bayes, multinomial logistic regression, and kNN.
- Comparing the performances of the classifiers: We considered not only the accuracy, the F1-score, and the AUC but also a weighted F1-score that better reflected the classification quality in the presence of a highly imbalanced dataset.
- Providing a per-case analysis of errors: This enables statisticians to make a more informed decision on the methods to use and exclude.
- Providing a operational time analysis for the methods with the best performances: This allows statisticians to select the most efficient methods.
- Providing an analysis of the classification performances: We also included the number of features generated by the embedding process.

We were aware that other embedding techniques (such as BERT [24]) and classification methods (such as LSTM [25], PIQN [26], and W2NER [27]) existed that would not be covered in our study. Some we had already tested, but more experimentation was necessary to obtain better results, while others were published only a short time before submitting this paper and could not, therefore, be considered in the present study. However, as far as we know, this was the most comprehensive study in the area of product classification for CPI computation, to date.

3. Data

The datasets were collected using a web-scraping technique of the main national e-commerce sites, and each record contained a product code provided by the retailer, the product name (which included a short description of the product), the price per unit, timestamps, and the retailer ID. We processed the data collection scripts on a weekly basis, with approximately 50,000 records collected each week. We used only the product names in our study in order to classify the products while ignoring the rest of the attributes. The samples used in our experiment included 2853 products from 15 classes corresponding to food and home appliance categories, and we manually labeled each product with its corresponding class.

The dataset was divided into a training (70%) and a testing set (30%). The records were randomly selected from the entire database, and they generally followed the same distribution of products as the initial dataset. In Table 1 and Figure 1, we present the distribution of the total number of products among the 15 selected classes, as well as the training and testing subsets.

Our dataset showed an important imbalance among the classes: 3 classes (05.3.1.1, 05.3.1.2, 05.3.1.3) contained 82.6% of the total number of products while the rest of the

12 classes contained only 17.4%. This imbalance between the size of the classes had consequences on the performance metrics of the classification methods, while the accuracy was generally accepted as a good performance indicator. In this case, high accuracy did not necessarily mean that the resulting classification was satisfactory since this high accuracy could be obtained only by correctly classifying items from larger classes. Therefore, in addition to the accuracy and the F1-score, we also used a weighted F1-score to report the performances of our classifiers. All the details of the metrics used are provided in the next section.

Table 1. The distribution of the selected products among the 15 ECOICOP classes.

ECOICOP Class Code	ECOICOP Class Name	Total No. Products	No. of Products in the Training Set	No. of Products in the Testing Set
01.1.1.2	Flours and other cereals	74	52	22
01.1.1.3	Bread	14	10	4
01.1.4.1	Fresh whole milk	45	31	14
01.1.4.2	Fresh low fat milk	36	25	11
01.1.4.7	Eggs	59	41	18
01.1.5.1	Butter	45	31	14
01.1.5.3	Olive oil	88	62	26
01.1.5.4	Other edible oils	42	29	13
01.1.6.1	Fresh or chilled fruit	17	12	5
01.1.7.3	Dried vegetables, other preserved or processed vegetables	21	15	6
01.1.7.4	Potatoes	21	15	6
01.1.8.1	Sugar	33	23	10
05.3.1.1	Refrigerators, freezers and fridge-freezers	931	652	279
05.3.1.2	Clothes washing machines, clothes drying machines and dish washing machines	767	537	230
05.3.1.3	Cookers	660	462	198

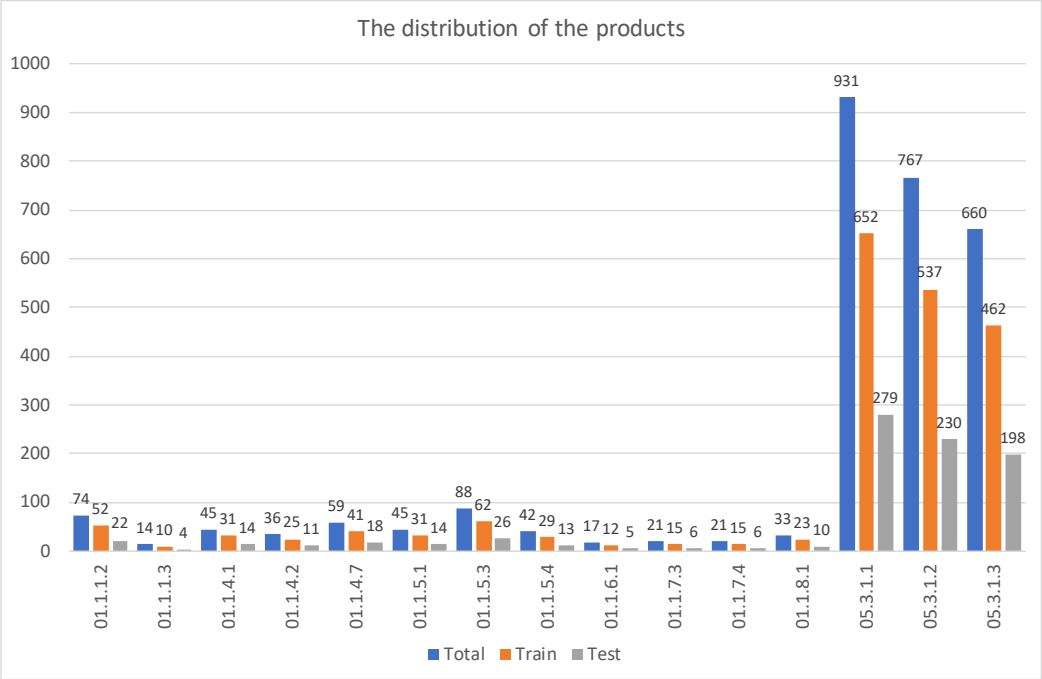

Figure 1. Distribution of the products among ECOICOP classes.

In Tables 2 and 3, we present a short statistical description of the initial dataset and the corresponding training and testing datasets.

Table 2. Descriptive statistics for the words in the dataset.

	Number of Words	Number of Unique Words	Average Word Length (in Chars) (Std. Dev.)	Min. Word Length (in Chars)	Max. Word Length (in Chars)
Entire dataset	41985	4536	4.86 (2.98)	1	24
Training dataset	29395	3669	4.86 (2.98)	1	24
Testing dataset	12590	2129	4.85 (2.99)	1	21

Table 3. Descriptive statistics for the product names.

	Average Number of Words (Std. Dev.)	Minimum Number of Words	Maximum Number of Words
Entire dataset	14.71 (6.34)	2	38
Training dataset	14.72 (6.42)	2	38
Testing dataset	14.71 (6.17)	2	34

4. Methods

Classification modeling approximates a mapping function (f) from input variables $\mathbf{X} = (x_1, x_2, \ldots, x_n)$, also called either predictors, features, or attributes, to a discrete output variable y, called the target or output variable. A classification model could be simply written as:

$$y = f(\mathbf{X}; \theta), \tag{1}$$

where $\mathbf{X} = (x_1, x_2, \ldots x_n)$ are the predictors; y is a categorical variable with two values (0/1, for example) for binary classification problems or a set of values in case of multi-class problems; and θ stands for a set of parameters. We used only supervised classification methods in our study. A supervised classification method started with a dataset consisting of pairs (y_i, X_i), where for each observation i, we knew the actual class (the value of y_i),

fit a model using these data, and then could predict the values of the output variables for unseen observations.

Therefore, in our case, y was the class of a product (with 15 different possible values for our particular dataset) while **X** was the name of the product. Because machine-learning-classification methods used numerical vectors as inputs, in order to be able to use any classification method, the first step was to transform the actual inputs (text data = the name of a product)into numeric values. Before applying the word-embedding techniques, we pre-processed our data by:

1. Tokenizing the product names;
2. Transforming all characters into lowercase characters;
3. Eliminating leading and trailing white spaces;
4. Trimming any unnecessary white spaces between words;
5. Eliminating punctuation marks, such as commas, semi-colons, and colons.

Transforming text data into numerical representation meant building a vector $X = (x_1, x_2, \ldots, x_n)$ with certain properties for each word. One important property of such a method would be to obtain similar embedding for similar words. Currently, there are several word-embedding techniques with different properties and characteristics. We selected the count vectorization [28], term-frequency—inverse-document-frequency [29], Word2Vec [30], FastText [31], and GloVe [32] methods to be used in our study.

In the following, we shortly describe each embedding method that was used in our study. For more details, an interested reader can consult the above cited references.

Count vectorization is very simple, and it involves counting the appearance of each word in a document. Suppose we have two products with the following names: P1, "white flour 000 for sponge cakes", and P2, "superior white flour from wheat 000". Count vectorization builds a vector representation of these two names by first making a set of unique words (the vocabulary of the problem) and then by assigning the number of appearances of each word in every document. An example with the these product names is presented in Table 4.

Table 4. Count vectorization.

Product Name	000	Cakes	Flour	For	From	Sponge	Superior	Wheat	White
P1	1	1	1	1	0	1	0	0	1
P2	1	0	1	0	1	0	1	1	1

Therefore, the first product name "white flour 000 for sponge cakes" has a vectorized form of $v1 = (1, 1, 1, 1, 0, 1, 0, 0, 1)$ and "superior white flour from wheat 000" of $v2 = (1, 0, 1, 0, 1, 0, 1, 1, 1)$.

This is a very simple and fast method of word vectorization, but it has some disadvantages. Firstly, different product names can have exactly the same vectorized representations since this method does not account for the order of the words. Secondly, there is no way to encode the context of the words. Thirdly, it cannot handle out-of-vocabulary words. Out-of-vocabulary words could appear in this context if a new product was presented to a classifier but the product name contained a word that was not in the training set. To mitigate this problem, one possible solution would be to use a very large vocabulary when building a dataset via word-embedding in order to exclude the chance of encountering new words.While for general text classification problems this could be a satisfactory solution, in our case, we had to handle words that might not exist in general vocabulary, since product names could contain words from other languages (especially English) or highly technical words. Rebuilding the vocabulary each time we presented the classifier with new product sets appeared to be the only acceptable solution in this specific case. For the first two problems, to ensure the awareness of the order and context of words, the count vectorization method would have to consider not only single words when building

vectorized representations, but also the sequences of consecutive words, called n-grams, where n is the number of words.

We implemented this vectorization method with the `superml` R package [33], using n-grams ranging from one to three words and removing the stop-words. The resulting vectors had more than 32,000 elements, which would be a serious issue for some of the machine-learning methods used for classification. Therefore, we limited the dimension of the embedding by considering only the first 3000 terms (single words and n-grams) ordered by their frequency. This value could be considered a parameter, and a search operation for the optimum value could be performed.

Term frequency—inverse document frequency (TF-IDF) was the second embedding method used in our study. TF-IDF builds upon a count factorization method by attributing more importance to certain words. Frequently used words in a text are considered less important since they are typically stop-words, and less common words are considered more important since they can carry useful information. The score of a word i in document j denoted by $w_{i,j}$ was given by:

$$w_{i,j} = tf_{i,j} \times idf_i, \tag{2}$$

where $tf_{i,j}$ is the frequency of word i in document j, and idf_i is determined by the following:

$$idf_i = \log \frac{n}{df_i} + 1, \tag{3}$$

where n is the total number of documents and df_i is the number of documents containing the word i. Therefore, the embedding for word i is given by $(w_{i,1}, w_{i,2}, \ldots w_{i,N})$, and N is the number of dimensions (we used 3000, as in the previous case).

This method had the same limitations as the previous one: unawareness of the context when vectorizing a word and the inability to build vectorizations for unknown words. The solution was the same as mentioned for the count factorization, i.e., using n-grams in addition to the individual words.

We implemented this method with the same `superml` R package, using single words, bi-grams, and tri-grams, and limiting the dimensions of the vectors to 3000.

Word2Vec is an algorithm that uses a set of words (a vocabulary, or a corpus) as input and produces a vectorized representation of each word as output, using a shallow neural network. There were two versions of this method: continuous bag of words (CBOW) and skip-gram.

The CBOW version of the algorithm attempted to guess a word w_i starting with the surrounding words $w_{i-m}, \ldots, w_{i-1}, w_{i+1}, \ldots w_{i+m}$ while the skip-gram version started from a word w_i and attempted to predict the surrounding words $w_{i-m}, \ldots, w_{i-1}, w_{i+1}, \ldots w_{i+m}$. Here, m is a parameter of the algorithm called the window size. The structures of the neural networks for both variants are depicted in Figure 2.

Consider, for example, the left side of the picture showing the skip-gram version. The algorithm started by building the vocabulary (or the corpus) of the problem and then encoding each word as a vector of the same dimension as the number of words in the vocabulary. The elements of this vector were all 0, except for an element on the position where the corresponding word appeared in the vocabulary, which had a value of 1. This simple method was called one-hot encoding. This vector was the input of the neural network. From the input to the hidden layer, the word vector was multiplied by a weight matrix W_1. The number of columns of this matrix, which was also the number of neurons in the hidden layer, would be the number of the features (the dimension) of the output. This was a hyper-parameter, and the performances of the algorithm could be tuned by testing different values for it. A second weight matrix W_2 was used to compute a score of each word, and using the $softmax$ function, the final output would be a vector with the posterior distribution of the words. The network was trained using a back-propagation algorithm.

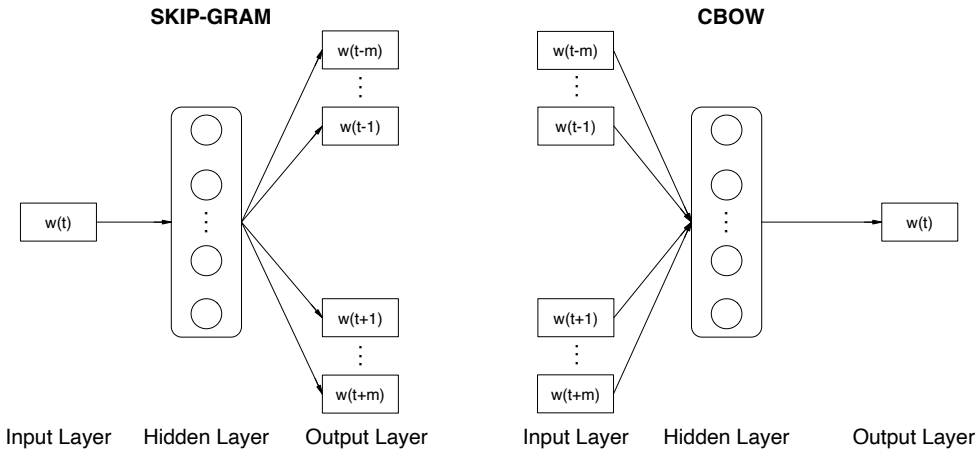

Figure 2. A schematic view of the neural networks in the Word2Vec method.

The Word2Vec algorithm provided the vectorized representation of each word, but for our problem, we needed a vectorized representation of the product name, which could be composed of several words. We used two methods to build these vectors: firstly, by adding the vectors of each word in the product name (ADD), and secondly by averaging these vectors (MEAN). We tested the classification methods with both versions. Therefore, for the Word2Vec method, we have four vectorizations for each product name: CBOW + ADD, CBOW + MEAN, skip-gram + ADD, and skip-gram + MEAN.

We implemented the Word2Vec vectorization using the `word2vec` R package [34], and we built vectors with 50 features. The number of features was limited to a small value in order to ensure that the operational time was acceptable for our experiment.

FastText builds on Word2Vec by involving not only the words but also the character n-grams (the sequences of n characters from a word). Therefore, this method could handle words not included in its vocabulary by attempting to build their embedding from the character n-grams used in the training process. It had the same two versions, CBOW and skip-gram, as Word2Vec. After obtaining the vectorization of each word, we proceeded to build the vectorization of the product names by following the original description of the algorithm: We divided each word embedding by its L2 norm and computed the average value of the word vectors in a product name for only those vectors with a non-zero L2.

We implemented this method using the `fastText` R package [35], and we set the dimension of the vectors at 50. We used word n-grams with up to three words, and character n-grams with n up to three to train the network.

GloVe goes a step further, and in addition to considering only local words for contextual information, it used word co-occurrence to integrate global information into the computations for word embedding.An element $m_{i,j}$ of the co-occurrence matrix indicated how many times a word w_i had co-occurred with word w_j. Given two words w_i and w_j and a third word, also called the probe word, w_k, GloVe used P_{eek}/P_j to compute the word embedding, where P_{ik} is the probability of seeing word w_i together with word w_k, which is simply computed by dividing the number of times words w_i and w_k appear together by the total number of times word w_i appears in the vocabulary. P_{jk} was computed in a similar way. Building the word-embedding was performed with a neural network, using a least-squares method, such as a log-bilinear cost function.

We implemented the GloVe method of vectorization using the `text2vec` R package [36], and we set the dimension of the vectors to 50.

Therefore, we built nine different vectorized representations for each product name, given by the following methods:

1. Count Vectorization;
2. TF-IDF

3. Word2Vec CBOW, with product name-embedding computed by adding each word-embedding—(Word2Vec CBOW ADD)
4. Word2Vec CBOW with product name-embedding computed by averaging each word-embedding—(Word2Vec CBOW MEAN)
5. Word2Vec skip-gram with product name-embedding computed by adding each word-embedding—(Word2Vec skip-gram ADD)
6. Word2Vec skip-gram with product name-embedding computed by averaging each word-embedding—(Word2Vec skip-gram MEAN)
7. FastText CBOW
8. FastText skip-gram
9. GloVe

Having the vectorized representations of the product names, we proceeded to apply several machine-learning-classification methods. We used a series of supervised classification methods, which are presented in Table 5, along with the implementation details.

We used classical methods, such as logistic regression, kNN, and multinomial naïve Bayes, all of which had surprisingly good results; basic decision trees (CART) and their more sophisticated variations (Bagged CART, C4.5, C50, random forests); and more modern methods such as support vector machines, artificial neural networks, and XGBoost. Most of the classification methods were used with all nine vectorizations of the product names with two exceptions. For the multinomial naïve Bayes, we used only the count vectorization and TF-IDF because it required only positive values for the features, and we excluded these two methods when using artificial neural networks because the software implementation did not support features with such a high dimensionality as produced by count vectorization and TF-IDF. For the tree-based methods, we included a repeated 10-fold cross-validation procedure because it was known that their results would have high variances.

A general data-flow diagram of the classification pipeline is presented in Figure 3.

Table 5. Machine learning classification methods.

Method	Software Implementation	Details
Multinomial Logistic Regression [37]	glmnetUtils package [38]	Applied for all 9 vectorization methods;
Multinomial Naïve Bayes [39]	naivebayes package [40]	Applied only for count vectorization and TF-IDF vectorization
Classification and Regression Trees (CART) [41]	`rpart` package [42]	Applied for all 9 vectorization methods; Applied with Gini and Information gain criteria to split the nodes
Bagged CART [43]	`e1071` [44] and `caret` [45] packages	Applied for all 9 vectorization methods Repeated 10-fold cross-validation to further reduce the variance
C4.5 [46]	`Rweka` package [47]	Applied for all 9 vectorization methods Repeated 10-fold cross-validation to further reduce the variance
C50 [48]	`C50` package [49]	Applied for all 9 vectorization methods Repeated 10-fold cross-validation to further reduce the variance
Random Forest [50]	`ranger` package [51]	Applied for all 9 vectorization methods Repeated 10-fold cross-validation to further reduce the variance
Support Vector Machines [52]	`e1071` [44] and `caret` [45] packages	Applied with *radial* and *sigmoid* kernels Applied for all 9 vectorization methods
Artificial Neural Networks [53]	`nnet` package [54]	One hidden layer Applied for Word2Vec, FastText and GloVe vectorization
kNN [55]	`caret` package [45]	Applied for all 9 vectorization methods
XGBoost [56]	`XGBoost` package [57]	Applied for all 9 vectorization methods

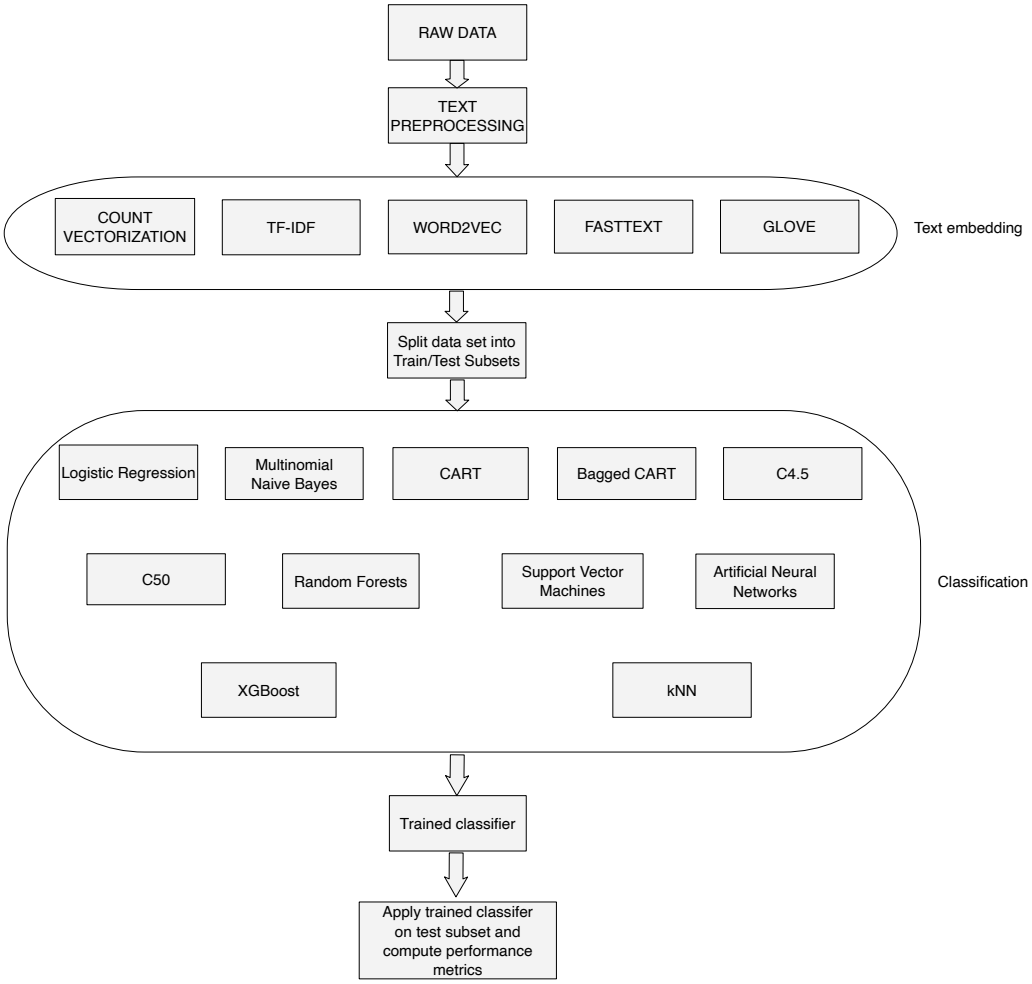

Figure 3. A schematic view classification pipeline.

5. Results

Our software was developed using the R package, and the scripts were available at https://github.com/bogdanoancea/autoencoder. We executed the data processing scripts on a desktop computer with an Intel Core i7-8559U processor at 4.5 GHz, 32 GB DDR4 RAM, and a Windows 11 operating system. The processing time of all classification methods was around 24 h.

We started with an exploratory data analysis to determine how well our classes were separated (or inter-leaved). Therefore, we built bi-dimensional visualizations of the dataset for all nine vectorization methods. We used the *t*-distributed stochastic neighbor-embedding (t-SNE) method [58] to reduce the dimensionality of the vectors from 3000 in the count vectorization and TF-IDF and from 50 for Word2Vec, FastText, and GloVe, to only 2. For the implementation, we used the `Rtsne` R package [59].

The bi-dimensional visualizations for all nine datasets are presented in Figure 4. We observed that the count vectorization, TF-IDF, FastText skip-gram, and even GloVe produced much better separations between product classes than Word2Vec, where there was significant interleaving, especially among smaller classes. Therefore, we expected to observe similar results when we applied the classification models and computed their individual performance metrics.

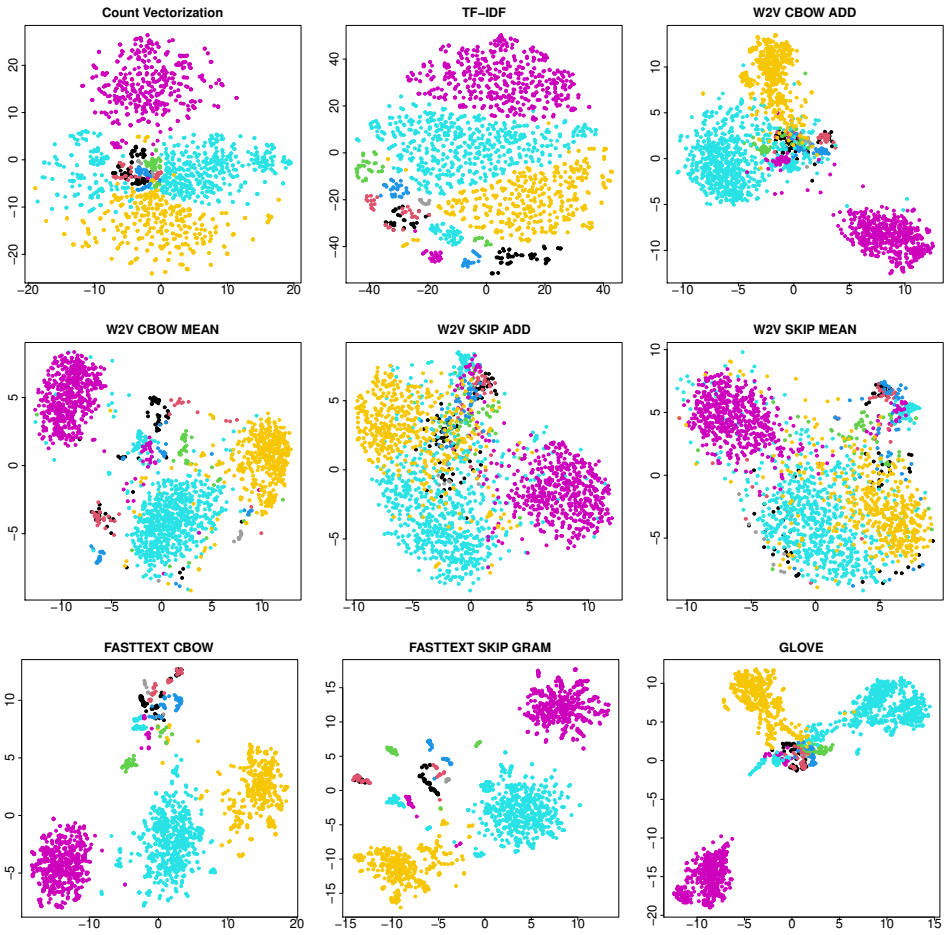

Figure 4. 2D visualization for the product name-embedding.

The performance metrics for the classification problems were derived from the well-known confusion matrix. In a two-class problem, we used the terms "positive" and "negative" for the two classes: We denoted the number of "positive" data points predicted correctly as TP (true-positive); the number of "negative" data points predicted correctly as TN (true-negative); the number of data points predicted in the "positive" class but belonging to the "negative" class as FP (false-positive); and the number of data points predicted in the "negative" class but belonging to the "positive" class as FN (false-negative). The performance metrics are shown in Table 6.

Table 6. Performance metrics for classification problems.

Metrics	Formula
Accuracy	$\frac{TP+TN}{P+N}$
Recall	$\frac{TP}{TP+FN}$
Precision	$Precision = \frac{TP}{TP+FP}$
F1-score	$\frac{TP}{TP+\frac{1}{2}\times(FP+FN)}$

Where $P = TP + FN$ and $N = FP + TN$.

In our case, using only the accuracy could be misleading because we could obtain very high accuracy if we predicted only the larger classes correctly; therefore, we used the F1-score in addition to the accuracy. In the case of a multi-class classification problem, we usually computed a per-class F1-score, and then we would report an aggregated form of these scores as the simple mean of the per-class F1-scores, called macro-F1. We also

computed a weighted macro-F1-score by defining the weight of class i as $w_i = \frac{N_i}{N}$, where N_i is the number of observations in class i and N is the total number of observations. To give more importance to small classes, we used the inverse of the weights defined as $v_i = \frac{1}{w_i}$ and $u_i = \frac{v_i}{\sum v_i}$.

Then, we defined the weighted macro-F1 as:

$$F1_w = \sum_{i=1}^{n} u_i \times F1_i, \tag{4}$$

where n is the number of classes and $F1_i$ is the F1-score for class i.

In addition to the accuracy and (weighted) F1-score, we also computed the multi-class AUC, as defined by [60], which was a mean of several individual AUCs and, therefore, could not be plotted.

The performance metrics for the all automatic classification and embedding methods are presented in Figure 5. In Tables 7 and 8, we listed all classification methods along with the embedding technique that provided the highest weighted F1-scores and accuracy values. An attempt to create a ranking according to the AUC would show similar results, but this metric was less sensitive, as the maximum value had the same values for the six classification methods: XGBoost, C50, C4.5, random forest, Bagged CART and support vector machines with a radial kernel. In Appendix A and in Tables A1–A13, we present the performance metrics for all the classification methods and all the embedding techniques. As shown in Tables 7 and 8, both the weighted F1-scores and accuracy values indicated that the best performing classification methods were logistic regression, the support vector machines with a radial kernel, the random forest combined with the FastText skip-gram embedding technique, and XGBoost combined with TF-IDF.

Table 7. Classification methods and embedding techniques with the highest weighted F1-scores.

Classification Method	Embedding Technique	Accuracy	F1	Weighted F1	AUC
Logistic Regression	FastText Skip-Gram	0.995	0.963	0.963	0.993
Support Vector Machines with Radial kernel	FastText Skip-Gram	0.994	0.972	0.957	0.983
Random Forest	FastText Skip-Gram	0.993	0.962	0.942	0.982
XGBoost	TF-IDF	0.992	0.966	0.934	0.998
kNN	FastText CBOW	0.989	0.943	0.931	0.969
C50	TF-IDF	0.992	0.963	0.929	0.997
C4.5	Count Vectorization	0.991	0.963	0.929	0.997
Bagged CART	Count Vectorization	0.992	0.958	0.919	0.997
Multinomial Naïve Bayes	Count Vectorization	0.991	0.955	0.915	0.996
Support Vector Machines with Sigmoid kernel	FastText Skip-Gram	0.980	0.930	0.799	0.977
CART-Gini Index	Count Vectorization	0.970	0.969	0.766	0.975
Artificial Neural Networks	GLOVE	0.961	0.840	0.869	0.906
CART-Information Gain	Count Vectorization	0.959	0.948	0.580	0.946

Table 8. Classification method—embedding technique with the highest accuracy.

Classification Method	Embedding Technique	Accuracy	F1	Weighted F1	AUC
Logistic Regression	FastText Skip-Gram	0.995	0.963	0.963	0.993
Support Vector Machines with Radial kernel	FastText Skip-Gram	0.994	0.972	0.957	0.983
Random Forest	FastText Skip-Gram	0.993	0.962	0.942	0.982
XGBoost	TF-IDF	0.992	0.966	0.934	0.998
C50	TF-IDF	0.992	0.963	0.929	0.997
Bagged CART	Count Vectorization	0.992	0.958	0.919	0.997
C4.5	Count Vectorization	0.991	0.963	0.929	0.997
Multinomial Naïve Bayes	Count Vectorization	0.991	0.955	0.915	0.996
kNN	FastText CBOW	0.989	0.943	0.931	0.969
Support Vector Machines with Sigmoid kernel	FastText Skip-Gram	0.980	0.930	0.799	0.977
CART-Gini Index	Count Vectorization	0.970	0.969	0.766	0.975
Artificial Neural Networks	GLOVE	0.961	0.840	0.869	0.906
CART-Information Gain	Count Vectorization	0.959	0.948	0.580	0.946

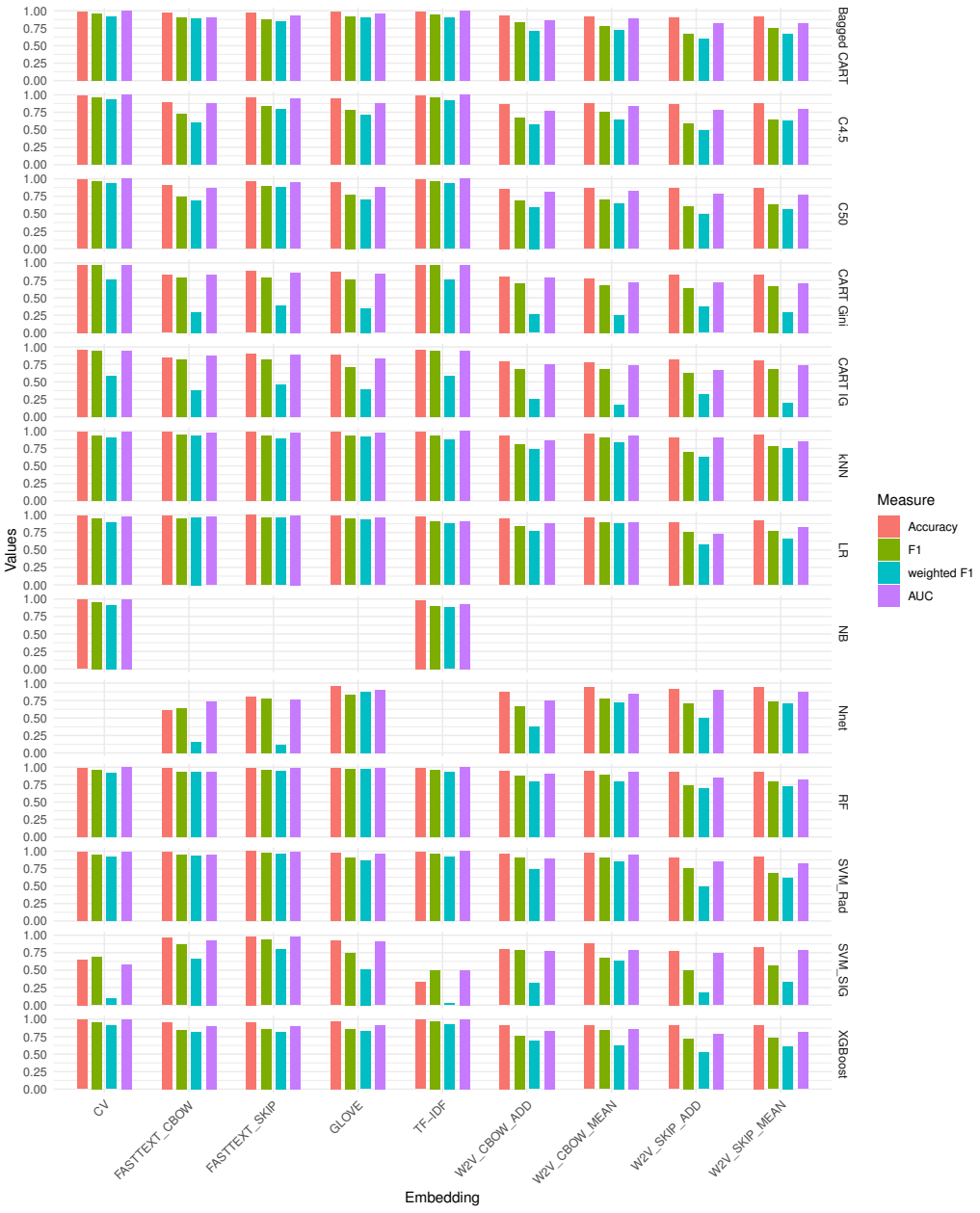

Figure 5. Performance metrics for automatic classification methods.

6. Discussion

The automatic product classification is a mandatory task when using big-data sources to complement classical data sources for consumer price statistics, as manual classification can be prohibitive in terms of the time needed for this task and the costs involved.

Our results showed very good classification performance with accuracy ranging from 0.326 to 0.995 and the weighted F1-score ranging from 0.095 to 0.963 for different word-embedding and classification combinations.

The best results in terms of the accuracy of the predictions were obtained for logistic regression at 0.995, support vector machines with a radial kernel at 0.994, and RF at 0.993 (all three classification methods combined with the FastText skip-gram word-embedding technique).

In terms of the weighted F1-score, the best classification methods were similar: logistic regression, support vector machines, and random forest, combined with the FastText skip-gram, with 0.963, 0.957, and 0.942, respectively.The AUC values also confirmed that these

three methods, combined with the FastText skip-gram embedding, had a very high power of distinction between the classes. At the same time, the lower values of the weighted F1 and AUC in the Word2Vec embedding (see Figure 5) showed that when using this embedding technique, the separation between classes was more difficult to obtain, regardless of the classification method.

Therefore, support vector machines using a radial kernel, logistic regression, and random forest, combined with the FastText skip-gram embedding technique, appeared to have the best results for our classification problem, regardless of the performance metrics used. They were followed by the XGBoost method combined with TF-IDF, which showed good results, as well, for both accuracy and weighted-F1 metrics. The kNN was the only classification method that provided a high accuracy and weighted F1-score, when combined with the FastText CBOW embedding, while almost all classification methods performed poorly with Word2Vec embedding. All tree-based methods showed the best results for count vectorization and TF-IDF embedding.

As a general conclusion of the results, we found the following:

- The FastText skip-gram, as well as the simple embedding methods, such as count vectorization and TF-IDF, yielded good results with the majority of classification methods, which was in line with the first visual inspection of the classes performed with t-SNE (see Figure 4). FastText had the advantage of being able to handle words not in the vocabulary, as well;
- The Word2Vec embedding had poor results for almost all classification methods. This was confirmed by both the t-SNE transformation and the performance metrics values;
- When analyzing how different classification methods performed using the same embedding techniques, we noted that the weighted F1 showed a much higher variability than the other metrics, and combined with a per-class error analysis, this confirmed our hypothesis that for highly imbalanced classes, the weighted F1 was a much better performance indicator than the accuracy or the simple macro-F1. The same conclusion held when analyzing how different embedding techniques performed for the same classification method. Very low values of the weighted F1-scores (for example, review the results of the CART with Gini or the information gain criteria for node splitting, support vector machines with a sigmoid kernel, and the artificial neural networks) were obtained even when the accuracy was high.
- Logistic regression, support vector machines, and random forest had good classification performances when they were combined with FastText skip-gram, count vectorization, and TF-IDF embedding techniques, while the same methods had weaker performances when combined with Word2Vec embedding;
- Surprisingly, even simple and old methods, such as logistic regression and naïve Bayes had good classification performances, with logistic regression showing the best values for the performance metrics and on the dataset considered in our case;
- Predictably, more elaborate decision tree-based methods (Bagged CART, C4.5, C50, random forest) performed better than the simple decision-tree-classification methods, with random forest being one of the best classifiers according to our results;
- The decision tree methods, with one exception (random forest), had the best results when combined with the count vectorization or TF-IDF embedding methods, potentially due to the higher dimensionality of the resulting embedding.

The results obtained in this experiment surpassed other recent approaches [16–19], which were already presented in a previous section.

Regarding the combinations between the classification and word embedding methods, in [17], the same combination (support vector machines + FastText, as in our study) was found to yield the best results, in terms of the F1-score.

For a more in-depth analysis of the performances of the classification methods, we selected the first three that showed the best performance metrics, namely logistic regression, support vector machines with a radial kernel, and random forest, all combined with the FastText skip-gram embedding, and computed the performance metrics for a varying

number of features generated during the text-vectorization process. Therefore, for logistic regression (LR), random forest (RF), and support vector machines (SVMs) with a radial kernel, we computed the accuracy, F1-score, and weighted F1-score, changing the number of features during the vectorization from 25 to 250, with a step-size of 5. The results are presented in Figure 6.

Figure 6. The performances of the classification versus the number of features.

All the performance metrics had an oscillating evolution with a general increasing trend, up to a maximum value, followed by an approximately constant value or even a slight decrease if we further increased the number of features. Table 9 shows the maximum values for the accuracy, F1-scores, and weighted F1-scores, along with the number of features.

Table 9. The maximum values of the performance metrics (all classification combined with FastText skip-gram).

Classification Method	Number of Features	Accuracy	F1	Weighted F1
LR	235	0.998	0.999	0.999
RF	145	0.998	0.999	0.999
SVMs with a Radial kernel	135	1	1	1

As shown, while the maximum values of the performance metrics had almost the same values for all three methods, the support vector machines with a radial kernel achieved the maximum classification performance with a lower number of features (135) than logistic regression (235) and random forest (145). To further analyze the performance of these three methods, we measured the execution time of each versus the number of features, and we presented the results in Figure 7.

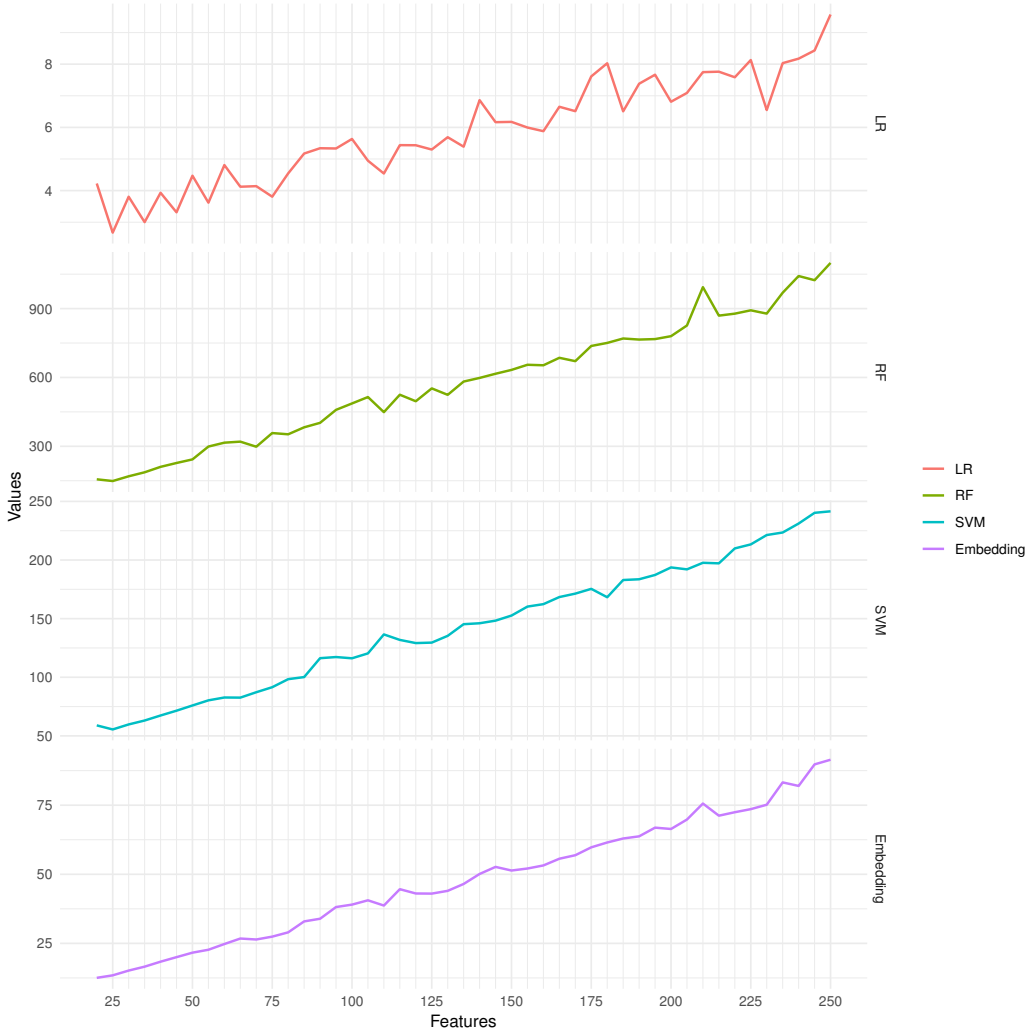

Figure 7. The execution time for the embedding process and the training time for logistic regression (LR), random forest (RF), and support vector machines (SVMs).

The time for the vectorization and the training time for all three classification methods showed a linear increasing trend but with different slopes. Fitting a simple linear regression model for the training time, we found the values of the slopes presented in Table 10.

Table 10. The slope of the execution time versus the number of features.

Classification Method	Slope Value	Std. Dev.
logistic regression	0.022	0.0011
random forest	3.84	0.0737
SVM with Radial kernel	0.79	0.0100

This indicated that the processing time for random forest and support vector machines increased rapidly with the number of features, though this increase was very small for logistic regression. Comparing also the absolute values of the running times of the embedding and the classification, we noted that the total processing time for logistic regression was almost constant, as compared to random forest and support vector machines. In Table 11, we present the values of these processing times for the number of features that generated the best performance metrics for each classification method. For logistic regression, the total time was dominated by the time needed by the embedding process, while for the other two methods, the total processing time was dominated by the training process.

Table 11. The processing time for embedding and training (all classification combined with FastText skip-gram).

Classification Method	No. Features	Embedding Time		Training Time		Total Time
		s	% of total	s	% of total	s
logistic regression	235	83.25	91.20%	8.04	8.8%	98.29
random forest	145	52.68	7.88%	616.15	92.12%	668.83
SVM with Radial kernel	135	46.51	24.25%	145.28	75.75%	191.77

Considering the total processing time, logistic regression provided the best performance with a total processing time two-fold less than support vector machines and almost seven-fold less than random forest, for the number of features that had the highest accuracy for each classification method. Most of the total time for logistic regression was spent on the vectorization of the product names (>90%), while for random forest and support vector machines, the situation was the opposite, where the training process was much longer than the vectorization. For random forest, even if the classification performances were very good, the processing time for larger datasets could be prohibitive for normal computing resources. This processing time analysis, along with the values for the accuracy and the F1-scores, which were almost the same for these three methods, recommended logistic regression as the most efficient classification method, followed by the support vector machines and random forest.

We conclude this section with a general process-flow diagram of the classification process. This example used the Word2Vec embedding method, and it is shown in Figure 8. The text pre-processing operations are shown in the upper part of the figure while the model fitting with the training set and the predictions with the testing set are shown in the lower part.

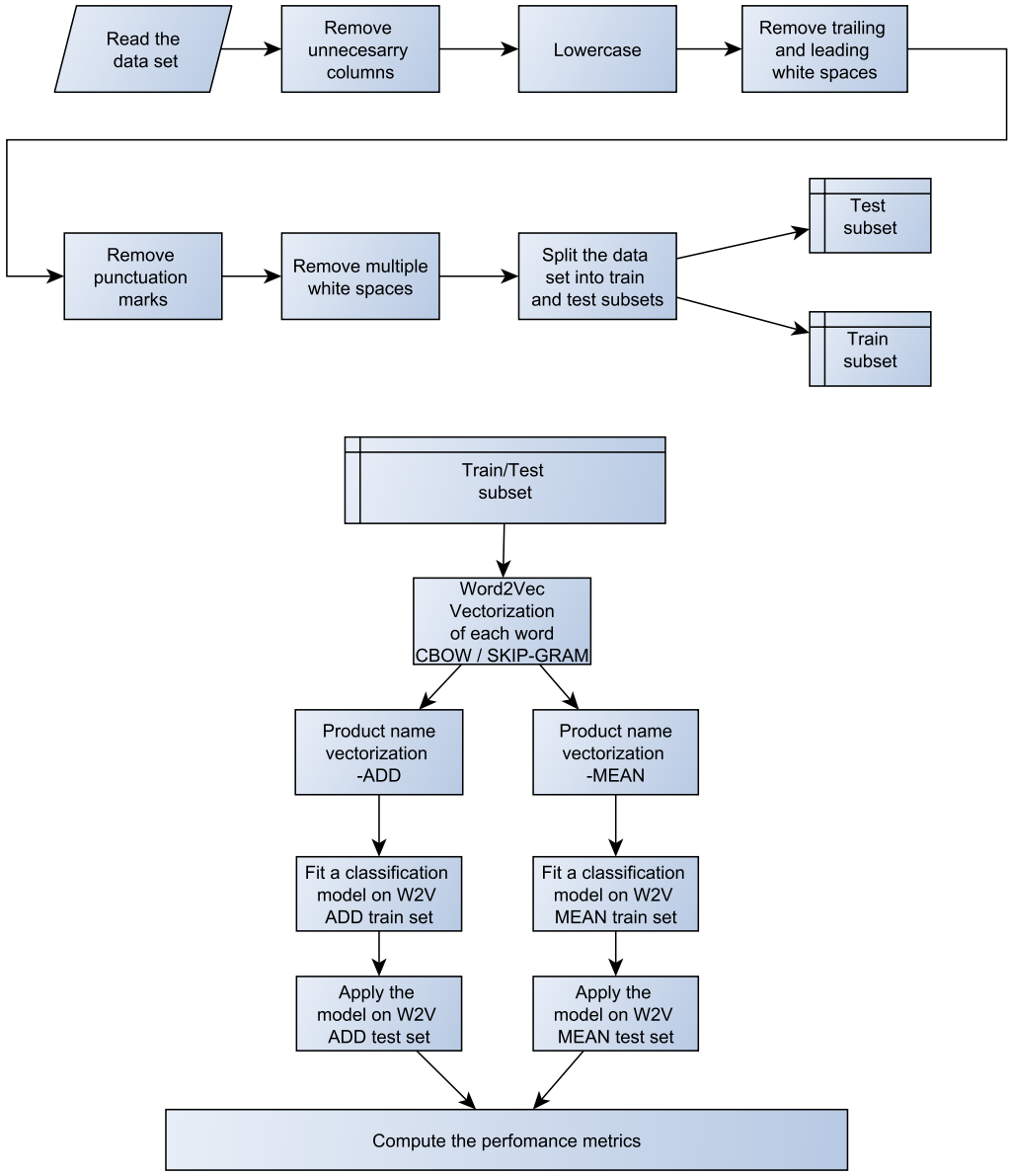

Figure 8. The classification process using Word2Vec embedding method.

7. Error Analysis

Despite these very good results, there were a number of factors to be considered further. The size of the dataset used in this study was rather small, and it is widely accepted that using larger datasets generally provides better classification performance. However, the time needed to execute the classifiers on larger datasets drastically increases, and special programming techniques should be used.

The distribution of the products among the classes was highly imbalanced, and this could have a negative impact on the quality of the classification results. Even if the accuracy of the classification was very high, a few errors in the smaller classes could have a significant impact on the final results of the price index. Therefore, the classification method should be chosen based on a metric that gives importance to smaller classes as well.

While the results could be considered good at a first glance, an error analysis could provide deeper insights into the performance and error sources. One first aspect that influenced the classification results was the composition of the training and testing sets. A simple verification showed that all 15 classes were present both in the training and testing datasets. We already presented the number of products in each class in Table 1. Therefore,

the generalization of the prediction models was not influenced by missing observations from training set, and the values of the performance metrics on the testing set were not influenced by some missing classes that could raise the values of the metrics artificially.

Next, the imbalance between the classes could have impacted the performance of the classifiers, and this was the reason we provided the weighted F1-scores alongside the F1-score and accuracy. Indeed, as shown in Tables A1–A13 and Figure 5, the values of the weighted F1-scores were less than the F1-scores and the accuracy.

To perform a more detailed error analysis, in Figures 9–11, we plotted the confusion matrices for the support vector machines with a radial kernel, random forest, and logistic regression classification models. Combined with the FastText skip-gram vectorization method, these three methods provided the best performance metrics, and we analyzed the errors on a per-case basis. We also presented the detailed performance metrics for each class separately for the combinations among the classification method and embedding techniques listed in Table 7, in Appendix B, and Tables A14–A26.

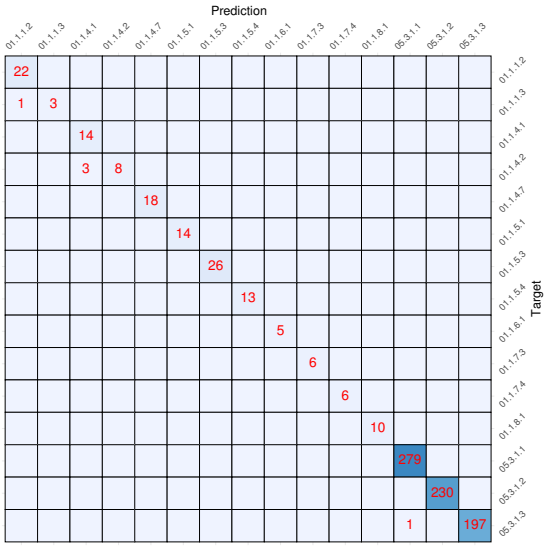

Figure 9. The confusion matrix for SVM with a radial kernel using the FastText skip-gram vectorization.

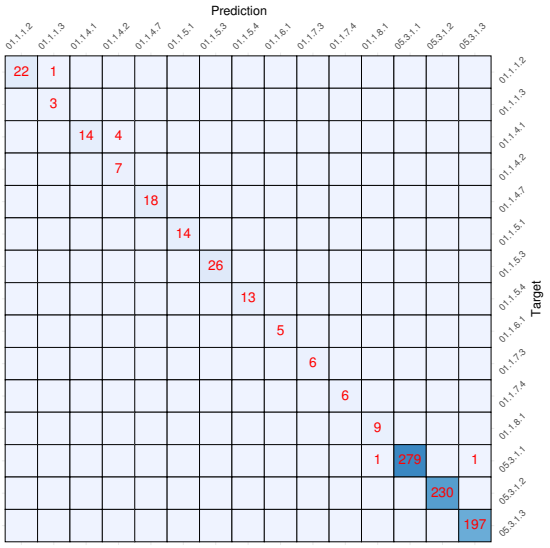

Figure 10. The confusion matrix for random forest using the FastText skip-gram vectorization.

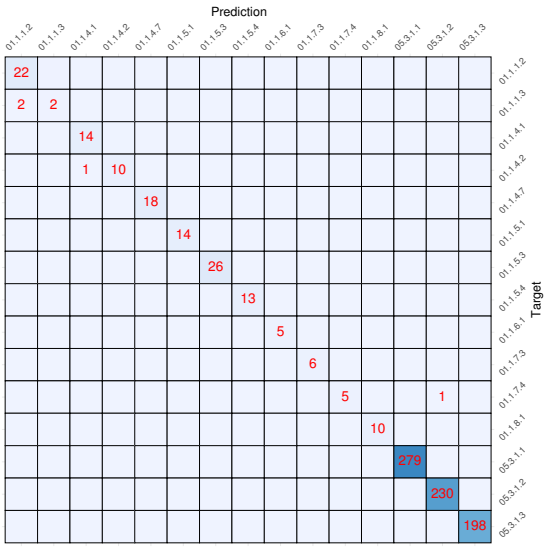

Figure 11. The confusion matrix for logistic regression using the FastText skip-gram vectorization.

One general remark was that the predictions for smaller classes performed well for support vector machines, random forest, and logistic regression, when combined with FastText skip-gram vectorization. On the test dataset, only five observations were incorrectly predicted by the support vector machines, four observations by logistic regression, and seven by random forest, out of which four were from smaller classes in the support vector machines, four in logistic regression, and five in random forest. Three of the observations belonging to the smaller classes were incorrectly predicted by the support-vector-machine model. These belonged to the class 01.1.4.2 (low-fat milk), but they were included in class 01.1.4.1 (whole milk); one belonged to the class 01.1.1.3 (bread), but it was included in the class 01.1.1.2 (flours and other cereals). This later observation was predicted incorrectly by logistic regression, as well, which incorrectly predicted another observation from the same class. An explanation could be related to these incorrectly predicted observations having names very similar to the observations in the class where they had been predicted, and thus the distance in the feature space could have been very small, resulting in them being predicted in the wrong class.

The other observation predicted incorrectly by the support-vector-machine model (from the larger classes) belonged to class 05.3.1.3 (cookers), but it was predicted as class 05.3.1.1 (refrigerators). In this case, the record had a brand name that was present with several records in the class where it had been incorrectly attributed, which could explain the error. Such particularities of the product names were found in random forest and logistic regression, as well. All observations from the larger classes were correctly predicted by logistic regression, but one observation from the 01.1.7.4 class was incorrectly included in the 05.3.1.2 class, one of the largest classes in our set. The random forest classifier predicted four observations incorrectly from low-fat milk (they were incorrectly included in the whole-milk class), and one observation from 01.1.1.3 (bread) class was incorrectly predicted as the class 01.1.1.2 (flours and other cereals), a situation similar to the support-vector-machine classifier. The same explanation could be applied here as well, as the classes were related one to each other (low-fat milk and whole milk, bread and cereals) with similar names that most likely produced vectorizations very close together in the feature space.

To conclude, we can state that these three classifiers had very good performances at the class level, even for the smaller classes, where incorrect classifications could affect the quality of the final results.

Further inspection of the per-class performance metrics revealed that both simple decision-tree methods (CART with the Gini index and CART with information gain used for node-splitting) had several classes missing from the predicted values, which indicated

they were less reliable for our purpose. This deficiency was solved in more sophisticated tree methods (C45, C50, Bagged CART), but there was still one class (01.4.1.2) where all these tree-based methods had a low accuracy of predictions. A low accuracy was noted also for the same class for the XGBoost method.

The support vector machines with a sigmoid kernel were also less reliable than the other methods, having one class of products missing from the predicted values, while kNN, neural networks, and multinomial naïve Bayes performed reasonably well at the class level.

When analyzing the per-class metrics, we also noticed that support vector machines, logistic regression, and random forest showed relatively good performances for all classes, the balanced accuracy for individual classes varying from 1 to 0.863 for the first method, from 1 to 0.75 for the second method, and from 1 to 0.888 for the final method, respectively.In contrast, the decision tree-based methods (CART, Bagged CART, C50, C4.5) showed a larger variation in the balanced accuracy between classes, in addition to the missing classes from the predictions. For all these methods, the lowest accuracy was recorded for the 01.1.4.2 class. However, these classifiers were not entirely incorrect when working with classes 01.1.4.1 and 01.1.4.2, as both were related to "milk". A better separation in the feature spaces of these two classes could improve the accuracy of the predictions.

The per-class error analysis confirmed that the support vector machines with a radial kernel, logistic regression, and random forest were the methods with the best results in our case and also assisted in identifying methods (simple decision trees, support vector machines with a sigmoid kernel) that could be excluded because they had produced predictions that made the CPI computations almost impossible.

8. Conclusions and Future Work

Currently, with the advent of the digital revolution, new data sources are being used to increase the timeliness and decrease the costs of the calculation process for several economic indicators used by policymakers throughout the world. One of these statistical indicators is the well-known CPI, computed in every country by the official statistics bureaus and used to fine-tune public policies. In addition to the classical methods for CPI computations, new data sources such as scanner data or web-scraped data have been used to either augment the way the CPI was computed or to compute entirely new price indices. Nevertheless, using such data sources had introduced a problem: Its large volume makes it almost impossible to manually classify the products according to the statistical methodologies in place. To solve this problem, automatic classification procedures that use machine-learning methods can be used. In this paper, we presented the results obtained after the experimentation with several automatic classification procedures: logistic regression, multinomial naïve Bayes, decision trees, bagged decision trees, C4.5, C50, random forest, support vector machines, artificial neural networks, kNN, and XGBoost. To our knowledge, this was one of the most comprehensive experiments in the area of product classifications, combining 9 different word-embedding techniques with 13 classification models.

We started with the transformation of the product names into numerical vectors, then we applied a series of machine-learning-classification methods. The results obtained were encouraging, as the methods tested showed very good performance.

The best results, both in terms of the accuracy and the weighted F1-scores, were obtained by logistic regression and support vector machines, followed by random forest, with the FastText skip-gram embedding technique. Using this embedding technique also provided the advantage of being able to treat words not already in the vocabulary. Regarding the embedding techniques, we noticed that all decision tree-based methods obtained good results with either the count vectorization or TF-IDF, which could have been associated with their much higher number of features (3000), as compared to Word2Vec, FastText, and GloVe, for which we generated only 50 features for each product name. A per-class error analysis showed that these methods performed poorly, having several classes entirely absent from the predictions on the test set. There was only one method, the artificial neural networks, that performed better with the GloVe embedding technique, and

kNN performed better with FastText CBOW. Surprisingly, the neural networks showed relatively poor results, as compared to the other methods, but we only used them with the default parameters. Choosing the optimum values for their parameters could greatly improve the classification results, but that would be a computationally intensive task that requires special programming techniques.

Nevertheless, these good results could also be explained by the structure of the product names, which did not vary significantly from one retailer to another, at least for the categories involved in this study. The "unseen" data used to test the performances of each method, i.e., the test subset, largely followed the same rules to build product names as the training set, and the classification consequently showed good results.

However, there were some issues to be considered in future work. Firstly, we conducted the classification algorithms on a relatively small dataset, yet the processing time was very high (approximately one day) when we used the repeated cross-validation procedure. The problem of computational complexity and high processing times will be even more acute when working with larger datasets. We envisage two solutions here: to use parallel programming techniques within the R software environment, or if the processing time is still high, to choose another language that could perform better, such as Python or even C++. With a faster execution, we could also use embedding with more dimensions than those used in this experiment.

Secondly, the machine-learning-classification methods have hyper-parameters that can greatly influence the results. To identify the optimum values, we intend to use a grid-search procedure, but only after we adopt another software environment. An experiment involving a grid search to choose the optimum value for the cost parameter of the support vector machines (with both kernels) and γ for the support vector machines with the radial kernel resulted in a processing time longer than 2 days, which we considered unacceptable for a pilot study.

Thirdly, there was the problem of the words not already in the vocabulary, i.e, words not present in the training set. While FastText could handle such words, the other embedding methods could not. One possible solution would be to build the vocabulary every time a new dataset needs to be classified, but this will likely result in a longer execution time.

Another direction for future research would be to use more complicated embedding techniques, such as BERT or one of its several variants, or other classification methods, such as LSTM or W2NER. Furthermore, finally, the implementation of an automatic procedure to rank the results and choose the best classification method should also be considered in future research.

Funding: This research received no external funding.

Data Availability Statement: The R scripts and the data used in this work are available at: https://github.com/bogdanoancea/autoencoder.

Conflicts of Interest: The author declares no conflicts of interest.

Appendix A. Performance Metrics for All Classification and Embedding Methods

Table A1. Performance metrics for logistic regression.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.981	0.943	0.893	0.969
TF-IDF	0.979	0.904	0.878	0.909
Word2Vec CBOW ADD	0.939	0.836	0.767	0.876
Word2Vec CBOW MEAN	0.963	0.886	0.881	0.891
Word2Vec SKIP ADD	0.897	0.748	0.574	0.720
Word2Vec SKIP MEAN	0.924	0.764	0.656	0.825
FastText CBOW	0.991	0.952	0.961	0.979
FastText skip-gram	0.995	0.963	0.962	0.994
GLOVE	0.982	0.946	0.929	0.965

Table A2. Performance metrics for the Multinomial naïve Bayes.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.991	0.955	0.915	0.996
TF-IDF	0.982	0.901	0.877	0.931

Table A3. Performance metrics for CART with Gini index for node splitting.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.969	0.968	0.766	0.974
TF-IDF	0.969	0.968	0.766	0.974
Word2Vec CBOW ADD	0.805	0.709	0.2581	0.788
Word2Vec CBOW MEAN	0.778	0.684	0.255	0.726
Word2Vec SKIP ADD	0.834	0.641	0.369	0.716
Word2Vec SKIP MEAN	0.825	0.665	0.291	0.710
FastText CBOW	0.827	0.788	0.291	0.830
FastText skip-gram	0.882	0.789	0.387	0.856
GLOVE	0.874	0.755	0.354	0.838

Table A4. Performance metrics for CART with Information Gain for node splitting.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.959	0.948	0.579	0.946
TF-IDF	0.959	0.948	0.579	0.946
Word2Vec CBOW ADD	0.787	0.687	0.257	0.751
Word2Vec CBOW MEAN	0.782	0.682	0.175	0.743
Word2Vec SKIP ADD	0.822	0.633	0.331	0.671
Word2Vec SKIP MEAN	0.805	0.689	0.204	0.736
FastText CBOW	0.850	0.815	0.385	0.872
FastText skip-gram	0.901	0.817	0.463	0.895
GLOVE	0.893	0.713	0.392	0.835

Table A5. Performance metrics for Bagged CART.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.992	0.958	0.919	0.997
TF-IDF	0.986	0.949	0.903	0.996
Word2Vec CBOW ADD	0.928	0.836	0.707	0.865
Word2Vec CBOW MEAN	0.924	0.774	0.725	0.888
Word2Vec SKIP ADD	0.899	0.668	0.596	0.816
Word2Vec SKIP MEAN	0.914	0.757	0.666	0.821
FastText CBOW	0.973	0.899	0.888	0.902
FastText skip-gram	0.968	0.869	0.844	0.926
GLOVE	0.980	0.924	0.899	0.965

Table A6. Performance metrics for C4.5.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.991	0.963	0.929	0.997
TF-IDF	0.989	0.958	0.919	0.997
Word2Vec CBOW ADD	0.861	0.662	0.575	0.759
Word2Vec CBOW MEAN	0.881	0.747	0.636	0.841
Word2Vec SKIP ADD	0.857	0.581	0.487	0.775
Word2Vec SKIP MEAN	0.870	0.645	0.628	0.794
FastText CBOW	0.895	0.717	0.597	0.876
FastText skip-gram	0.952	0.831	0.797	0.946
GLOVE	0.942	0.783	0.707	0.879

Table A7. Performance metrics for C50.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.991	0.963	0.929	0.997
TF-IDF	0.992	0.963	0.929	0.997
Word2Vec CBOW ADD	0.859	0.685	0.594	0.807
Word2Vec CBOW MEAN	0.868	0.698	0.643	0.823
Word2Vec SKIP ADD	0.871	0.605	0.487	0.786
Word2Vec SKIP MEAN	0.859	0.633	0.561	0.769
FastText CBOW	0.901	0.739	0.688	0.864
FastText skip-gram	0.960	0.895	0.873	0.946
GLOVE	0.943	0.774	0.696	0.878

Table A8. Performance metrics for random forest.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.992	0.958	0.919	0.997
TF-IDF	0.992	0.962	0.924	0.997
Word2Vec CBOW ADD	0.951	0.869	0.799	0.9042
Word2Vec CBOW MEAN	0.946	0.887	0.792	0.9288
Word2Vec SKIP ADD	0.929	0.743	0.691	0.853
Word2Vec SKIP MEAN	0.925	0.796	0.719	0.825
FastText CBOW	0.987	0.933	0.935	0.935
FastText skip-gram	0.993	0.962	0.942	0.982
GLOVE	0.991	0.968	0.934	0.993

Table A9. Performance metrics for support vector machines with Sigmoid kernel.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.649	0.682	0.095	0.579
TF-IDF	0.326	0.492	0.021	0.500
Word2Vec CBOW ADD	0.798	0.779	0.318	0.769
Word2Vec CBOW MEAN	0.876	0.675	0.633	0.781
Word2Vec SKIP ADD	0.773	0.488	0.175	0.739
Word2Vec SKIP MEAN	0.822	0.562	0.326	0.784
FastText CBOW	0.958	0.861	0.662	0.927
FastText skip-gram	0.980	0.930	0.799	0.977
GLOVE	0.9217	0.746	0.512	0.905

Table A10. Performance metrics for support vector machines with Radial kernel.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.991	0.949	0.911	0.991
TF-IDF	0.992	0.958	0.919	0.997
Word2Vec CBOW ADD	0.953	0.906	0.743	0.891
Word2Vec CBOW MEAN	0.966	0.902	0.843	0.939
Word2Vec SKIP ADD	0.909	0.748	0.484	0.849
Word2Vec SKIP MEAN	0.919	0.682	0.608	0.816
FastText CBOW	0.991	0.941	0.930	0.950
FastText skip-gram	0.994	0.972	0.957	0.983
GLOVE	0.976	0.901	0.857	0.965

Table A11. Performance metrics for Neural networks.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Word2Vec CBOW ADD	0.874	0.665	0.383	0.758
Word2Vec CBOW MEAN	0.947	0.780	0.727	0.850
Word2Vec SKIP ADD	0.919	0.717	0.503	0.904
Word2Vec SKIP MEAN	0.945	0.738	0.718	0.872
FastText CBOW	0.619	0.642	0.159	0.734
FastText skip-gram	0.803	0.777	0.117	0.759
GLOVE	0.961	0.839	0.869	0.906

Table A12. Performance metrics for XGBoost.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.991	0.954	0.909	0.996
TF-IDF	0.992	0.966	0.934	0.997
Word2Vec CBOW ADD	0.911	0.769	0.689	0.833
Word2Vec CBOW MEAN	0.910	0.839	0.631	0.857
Word2Vec SKIP ADD	0.910	0.725	0.523	0.787
Word2Vec SKIP MEAN	0.917	0.728	0.611	0.812
FastText CBOW	0.963	0.846	0.819	0.901
FastText skip-gram	0.963	0.860	0.824	0.902
GLOVE	0.966	0.864	0.826	0.918

Table A13. Performance metrics for kNN.

Embedding Technique	Accuracy	F1	Weighted F1	AUC
Count Vectorization	0.987	0.933	0.902	0.981
TF-IDF	0.986	0.933	0.876	0.994
Word2Vec CBOW ADD	0.931	0.806	0.731	0.867
Word2Vec CBOW MEAN	0.952	0.898	0.841	0.937
Word2Vec SKIP ADD	0.902	0.689	0.629	0.899
Word2Vec SKIP MEAN	0.938	0.784	0.747	0.849
FastText CBOW	0.989	0.943	0.931	0.969
FastText skip-gram	0.987	0.936	0.895	0.979
GLOVE	0.984	0.929	0.914	0.973

Appendix B. Per Class Performance Metrics for Classification and Embedding Methods

For the performance metrics listed below we used the standard definitions, see for example [61].

Table A14. Performance metrics for support vector machines with Radial kernel with FastText SKIP GRAM vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	1.000	0.998	0.956	1.000	0.956	1.000	0.977	0.025	0.025	0.026	0.999
01.1.1.3	0.750	1.000	1.000	0.998	1.000	0.750	0.857	0.004	0.003	0.003	0.875
01.1.4.1	1.000	0.996	0.823	1.000	0.823	1.000	0.903	0.016	0.016	0.019	0.998
01.1.4.2	0.727	1.000	1.000	0.996	1.000	0.727	0.842	0.012	0.009	0.009	0.863
01.1.4.7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.021	0.021	0.021	1.000
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.005	0.005	0.005	1.000
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.8.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.011	0.011	0.011	1.000
05.3.1.1	1.000	0.998	0.996	1.000	0.996	1.000	0.998	0.325	0.325	0.327	0.999
05.3.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.268	0.268	0.268	1.000
05.3.1.3	0.994	1.000	1.000	0.998	1.000	0.994	0.997	0.231	0.230	0.230	0.997

Table A15. Performance metrics for Logistic regression with FastText SKIP GRAM vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	1.000	0.997	0.916	1.000	0.916	1.000	0.956	0.025	0.025	0.028	0.998
01.1.1.3	0.500	1.000	1.000	0.997	1.000	0.500	0.666	0.004	0.002	0.002	0.750
01.1.4.1	1.000	0.998	0.933	1.000	0.933	1.000	0.965	0.016	0.016	0.017	0.999
01.1.4.2	0.909	1.000	1.000	0.998	1.000	0.909	0.952	0.012	0.011	0.011	0.954
01.1.4.7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.021	0.021	0.021	1.000
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.005	0.005	0.005	1.000
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	0.833	1.000	1.000	0.998	1.000	0.833	0.909	0.007	0.005	0.005	0.916
01.1.8.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.011	0.011	0.011	1.000
05.3.1.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.325	0.325	0.325	1.000
05.3.1.2	1.000	0.998	0.995	1.000	0.995	1.000	0.997	0.268	0.268	0.269	0.999
05.3.1.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.231	0.231	0.231	1.000

Table A16. Performance metrics for random forest with FastText SKIP GRAM vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	0.956	1.000	1.000	0.998	1.000	0.956	0.977	0.026	0.025	0.025	0.978
01.1.1.3	1.000	0.998	0.750	1.000	0.750	1.000	0.857	0.003	0.003	0.004	0.999
01.1.4.1	0.777	1.000	1.000	0.995	1.000	0.777	0.875	0.021	0.016	0.016	0.888
01.1.4.2	1.000	0.995	0.636	1.000	0.636	1.000	0.777	0.008	0.008	0.012	0.997
01.1.4.7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.021	0.021	0.021	1.000
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.005	0.005	0.005	1.000
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.8.1	1.000	0.998	0.900	1.000	0.900	1.000	0.947	0.010	0.010	0.011	0.999
05.3.1.1	0.992	1.000	1.000	0.996	1.000	0.992	0.996	0.328	0.325	0.325	0.996
05.3.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.268	0.268	0.268	1.000
05.3.1.3	1.000	0.998	0.994	1.000	0.994	1.000	0.997	0.230	0.230	0.231	0.999

Table A17. Performance metrics for kNN with FastText-CBOW vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	0.954	0.998	0.954	0.998	0.954	0.954	0.954	0.025	0.024	0.025	0.970
01.1.1.3	0.750	1.000	1.000	0.998	1.000	0.750	0.857	0.004	0.003	0.003	0.875
01.1.4.1	0.928	0.996	0.812	0.998	0.812	0.928	0.866	0.016	0.015	0.018	0.962
01.1.4.2	0.727	0.998	0.888	0.996	0.888	0.727	0.800	0.012	0.009	0.010	0.863
01.1.4.7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.021	0.021	0.021	1.000
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	1.000	0.998	0.833	1.000	0.833	1.000	0.909	0.005	0.005	0.007	0.999
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	0.833	1.000	1.000	0.998	1.000	0.833	0.909	0.007	0.005	0.005	0.916
01.1.8.1	0.900	0.997	0.818	0.998	0.818	0.900	0.857	0.011	0.010	0.012	0.948
05.3.1.1	1.000	0.998	0.996	1.000	0.996	1.000	0.998	0.325	0.325	0.327	0.999
05.3.1.2	0.995	1.000	1.000	0.998	1.000	0.995	0.997	0.268	0.267	0.267	0.999
05.3.1.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.231	0.231	0.231	1.000

Table A18. Performance metrics for C50 with TF-IDF vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.025	0.025	0.025	1.000
01.1.1.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.004	0.004	0.004	1.000
01.1.4.1	1.000	0.992	0.700	1.000	0.700	1.000	0.823	0.016	0.016	0.023	0.996
01.1.4.2	0.454	1.000	1.000	0.992	1.000	0.454	0.625	0.012	0.005	0.005	0.727
01.1.4.7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.021	0.021	0.021	1.000
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.005	0.005	0.005	1.000
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.8.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.011	0.011	0.011	1.000
05.3.1.1	1.000	0.998	0.996	1.000	0.996	1.000	0.998	0.325	0.325	0.327	0.999
05.3.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.268	0.268	0.268	1.000
05.3.1.3	0.994	1.000	1.000	0.998	1.000	0.994	0.997	0.231	0.230	0.230	0.997

Table A19. Performance metrics for Bagged CART with CV vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.025	0.025	0.025	1.000
01.1.1.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.004	0.004	0.004	1.000
01.1.4.1	0.928	0.992	0.684	0.998	0.684	0.928	0.787	0.016	0.015	0.022	0.960
01.1.4.2	0.454	0.998	0.833	0.992	0.833	0.454	0.588	0.012	0.005	0.007	0.726
01.1.4.7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.021	0.021	0.021	1.000
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.005	0.005	0.005	1.000
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.8.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.011	0.011	0.011	1.000
05.3.1.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.325	0.325	0.325	1.000
05.3.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.268	0.268	0.268	1.000
05.3.1.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.231	0.231	0.231	1.000

Table A20. Performance metrics for C45 with CV vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.025	0.025	0.025	1.000
01.1.1.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.004	0.004	0.004	1.000
01.1.4.1	1.000	0.992	0.700	1.000	0.700	1.000	0.823	0.016	0.016	0.023	0.996
01.1.4.2	0.454	1.000	1.000	0.992	1.000	0.454	0.625	0.012	0.005	0.005	0.727
01.1.4.7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.021	0.021	0.021	1.000
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.005	0.005	0.005	1.000
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.8.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.011	0.011	0.011	1.000
05.3.1.1	1.000	0.996	0.992	1.000	0.992	1.000	0.996	0.325	0.325	0.328	0.998
05.3.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.268	0.268	0.268	1.000
05.3.1.3	0.989	1.000	1.000	0.996	1.000	0.989	0.994	0.231	0.228	0.228	0.994

Table A21. Performance metrics for Multinomial naïve Bayes with CV vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.025	0.025	0.025	1.000
01.1.1.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.004	0.004	0.004	1.000
01.1.4.1	0.928	0.992	0.684	0.998	0.684	0.928	0.787	0.016	0.015	0.022	0.960
01.1.4.2	0.454	0.998	0.833	0.992	0.833	0.454	0.588	0.012	0.005	0.007	0.726
01.1.4.7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.021	0.021	0.021	1.000
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.005	0.005	0.005	1.000
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.8.1	0.900	1.000	1.000	0.998	1.000	0.900	0.947	0.011	0.010	0.010	0.950
05.3.1.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.325	0.325	0.325	1.000
05.3.1.2	1.000	0.998	0.995	1.000	0.995	1.000	0.997	0.268	0.268	0.269	0.999
05.3.1.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.231	0.231	0.231	1.000

Table A22. Performance metrics for XGBoost with TF-IDF vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.025	0.025	0.025	1.000
01.1.1.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.004	0.004	0.004	1.000
01.1.4.1	0.857	0.995	0.750	0.997	0.750	0.857	0.800	0.016	0.014	0.021	0.926
01.1.4.2	0.636	0.997	0.777	0.995	0.777	0.636	0.700	0.012	0.005	0.008	0.817
01.1.4.7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.021	0.021	0.021	1.000
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.005	0.005	0.005	1.000
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.8.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.011	0.011	0.011	1.000
05.3.1.1	1.000	0.998	0.996	1.000	0.996	1.000	0.998	0.325	0.325	0.325	0.999
05.3.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.268	0.268	0.268	1.000
05.3.1.3	0.995	1.000	1.000	0.998	1.000	0.995	0.997	0.231	0.231	0.231	0.997

Table A23. Performance metrics for support vector machines with Sigmoid with FastText skip-gram vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	1.000	0.995	0.846	1.000	0.846	1.000	0.916	0.025	0.025	0.030	0.997
01.1.1.3	0.250	1.000	1.000	0.996	1.000	0.250	0.400	0.004	0.001	0.001	0.625
01.1.4.1	1.000	0.986	0.560	1.000	0.560	1.000	0.717	0.016	0.016	0.029	0.993
01.1.4.2	0.000	1.000	NaN	0.987	NA	0.000	NA	0.012	0.000	0.000	0.500
01.1.4.7	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.021	0.021	0.021	1.000
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.005	0.005	0.005	1.000
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.8.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.011	0.011	0.011	1.000
05.3.1.1	0.992	1.000	1.000	0.996	1.000	0.992	0.996	0.325	0.323	0.323	0.996
05.3.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.268	0.268	0.268	1.000
05.3.1.3	0.994	0.996	0.989	0.998	0.989	0.994	0.992	0.231	0.230	0.232	0.995

Table A24. Performance metrics for CART with Information Gain with CV vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	0.863	0.992	0.760	0.996	0.760	0.863	0.808	0.025	0.022	0.029	0.928
01.1.1.3	0.000	1.000	NaN	0.995	NA	0.000	NA	0.004	0.000	0.000	0.500
01.1.4.1	1.000	0.986	0.560	1.000	0.560	1.000	0.717	0.016	0.016	0.029	0.993
01.1.4.2	0.000	1.000	NaN	0.987	NA	0.000	NA	0.012	0.000	0.000	0.500
01.1.4.7	0.888	1.000	1.000	0.997	1.000	0.888	0.941	0.021	0.018	0.018	0.944
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	0.000	1.000	NaN	0.994	NA	0.000	NA	0.005	0.000	0.000	0.500
01.1.7.3	0.000	1.000	NaN	0.992	NA	0.000	NA	0.007	0.000	0.000	0.500
01.1.7.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.8.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.011	0.011	0.011	1.000
05.3.1.1	1.000	0.968	0.939	1.000	0.939	1.000	0.968	0.325	0.325	0.346	0.984
05.3.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.268	0.268	0.268	1.000
05.3.1.3	0.979	1.000	1.000	0.993	1.000	0.979	0.989	0.231	0.226	0.226	0.989

Table A25. Performance metrics for CART with Gini index with CV vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.025	0.025	0.025	1.000
01.1.1.3	0.000	1.000	NaN	0.995	NA	0.000	NA	0.004	0.000	0.000	0.500
01.1.4.1	1.000	0.986	0.560	1.000	0.560	1.000	0.717	0.016	0.016	0.029	0.993
01.1.4.2	0.000	1.000	NaN	0.987	NA	0.000	NA	0.012	0.000	0.000	0.500
01.1.4.7	0.888	1.000	1.000	0.997	1.000	0.888	0.941	0.021	0.018	0.018	0.944
01.1.5.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.016	0.016	0.016	1.000
01.1.5.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.030	0.030	0.030	1.000
01.1.5.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.015	0.015	0.015	1.000
01.1.6.1	0.000	1.000	NaN	0.994	NA	0.000	NA	0.005	0.000	0.000	0.500
01.1.7.3	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.7.4	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.007	0.007	0.007	1.000
01.1.8.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.011	0.011	0.011	1.000
05.3.1.1	1.000	0.974	0.948	1.000	0.948	1.000	0.973	0.325	0.325	0.343	0.987
05.3.1.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.268	0.268	0.268	1.000
05.3.1.3	0.979	1.000	1.000	0.993	1.000	0.979	0.989	0.231	0.226	0.226	0.989

Table A26. Performance metrics for Neural Networks with GLOVE vectorization at class level.

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
01.1.1.2	0.818	0.994	0.782	0.995	0.782	0.818	0.800	0.025	0.021	0.026	0.906
01.1.1.3	0.500	0.996	0.400	0.997	0.400	0.500	0.444	0.004	0.002	0.005	0.748
01.1.4.1	1.000	0.998	0.933	1.000	0.933	1.000	0.965	0.016	0.016	0.017	0.999
01.1.4.2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.012	0.012	0.012	1.000
01.1.4.7	0.944	0.998	0.944	0.998	0.944	0.944	0.944	0.021	0.019	0.021	0.971
01.1.5.1	0.857	1.000	1.000	0.997	1.000	0.857	0.923	0.016	0.014	0.014	0.928
01.1.5.3	1.000	0.997	0.928	1.000	0.928	1.000	0.962	0.030	0.030	0.032	0.998
01.1.5.4	0.769	0.998	0.909	0.996	0.909	0.769	0.833	0.015	0.011	0.012	0.884
01.1.6.1	0.400	0.997	0.500	0.996	0.500	0.400	0.444	0.005	0.002	0.004	0.698
01.1.7.3	0.833	0.997	0.714	0.998	0.714	0.833	0.769	0.007	0.005	0.008	0.915
01.1.7.4	0.666	0.998	0.800	0.997	0.800	0.666	0.727	0.007	0.004	0.005	0.832
01.1.8.1	0.800	0.998	0.888	0.997	0.888	0.800	0.842	0.011	0.009	0.010	0.899
05.3.1.1	0.993	0.996	0.985	0.996	0.985	0.992	0.989	0.325	0.323	0.328	0.992
05.3.1.2	0.969	0.993	0.982	0.988	0.982	0.969	0.975	0.268	0.260	0.265	0.981
05.3.1.3	0.979	0.991	0.970	0.993	0.970	0.979	0.974	0.231	0.226	0.233	0.985

References

1. Harchaoui, T. M.; Janssen, R.V. How can big data enhance the timeliness of official statistics?: The case of the U.S. consumer price index. *Int. J. Forecast.* **2018**, *4392*, 225–234. [CrossRef]
2. Ivancic, L.; Erwin Diewert, W.; Fox, K.J. Scanner data, time aggregation and the construction of price indexes. *J. Econom.* **2011**, *161*, 24–35. [CrossRef]
3. Macias, P.; Stelmasiak, D.; Szafranek, K. Nowcasting food inflation with a massive amount of online prices. *Int. J. Forecast.* **2022**, *39*, 809–826. [CrossRef]
4. Yim, S.T.; Son, J.C.; Lee, J. Spread of E-commerce, prices and inflation dynamics: Evidence from online price big data in Korea. *J. Asian Econ.* **2022**, *80*, 101475. [CrossRef]
5. De Haan, J.; van der Grient, H.A. Eliminating chain drift in price indexes based on scanner data. *J. Econom.* **2011**, *161*, 36–46. [CrossRef]

6. Cavallo, A.; Rigobon, R. The Billion Prices Project: Using Online Prices for Inflation Measurement and Research. *J. Econ. Perspect.* **2016**, *30*, 151–178. [CrossRef]
7. Abe, N.; Shinozaki, K. *Compilation of Experimental Price Indices Using big data and Machine Learning: A Comparative Analysis and Validity Verification of Quality Adjustments*; Bank of Japan Working Paper Series, 18-E-13; Bank of Japan: Tokyo, Japan, 2018.
8. Oancea, B.; Necula, M. Web Scraping Techniques for Price Statistics—The Romanian Experience. *J. IAOS* **2019**, *35*, 657–667. [CrossRef]
9. Wankhade, M.; Rao, A.C.S.; Kulkarni, C. A survey on sentiment analysis methods, applications, and challenges. *Artifficial Intell. Rev.* **2022**, *55*, 5731–5780. [CrossRef]
10. Van den Bulk, L.M.; Bouzembrak, Y.; Gavai, A.; Liu, N.; van den Heuvel, L.J.; Marvin, H.J.P. Automatic classification of literature in systematic reviews on food safety using machine learning. *Curr. Res. Food Sci.* **2022**, *5*, 84–95. [CrossRef]
11. Santos, T.; Tariq, A.; Gichoya, J.W.; Trivedi, H.; Banerjee, I. Automatic Classification of Cancer Pathology Reports: A Systematic Review. *J. Pathol. Inform.* **2022**, *13*, 100003. [CrossRef]
12. Blanz, V.; Scholokopf, B .,; Bulthoff, H.; Burges, C.; Vapnik, V.N.; Vetter, V. Comparison of view-based object recognition algorithms using realistic 3D models. In Proceedings of the International Conference on Artificial Neural Networks—ICNN96, Berlin, Germany, 16–19 July 1996.
13. Calainho, F.D.; van de Minne, A.M.; Francke, M.K. A Machine Learning Approach to Price Indices: Applications in Commercial Real Estate. *J. Real Estate Financ. Econ.* **2022**. [CrossRef]
14. RAMON—Reference and Management of Nomenclatures. Available online: https://ec.europa.eu/eurostat/ramon/nomenclatures/index.cfm?TargetUrl=\LST_NOM_DTL&StrNom=COICOP_2018&StrLanguageCode=EN&IntPcKey=&StrLayoutCode=HIERARCHIC (accessed on 10 August 2022).
15. Roberson, A. Automatic Product Categorization for Official Statistics. In Proceedings of the 2019 Workshop on Widening NLP, Florence, Italy, 28 July 2019; pp. 68–72.
16. Roberson, A. Applying Machine Learning for Automatic Product Categorization. *J. Off. Stat.* **2021**, *37*, 395—410. [CrossRef]
17. Martindale, H.; Rowland, E.; Flower, T.; Clews, G. Semi-supervised machine learning with word embedding for classification in price statistics. *Data Policy* **2020**, *2*, e12. [CrossRef]
18. Muller, D.M. Classification of Consumer Goods into 5-Digit COICOP 2018 Codes. Master's Thesis, Norwegian University of Life Sciences, As, Norway, December 2021.
19. Myklatun, K.H. Using Machine Learning in the Consumer Price Index. In Proceedings of the Nordic Statistical Meeting, Helsinki, Finland, 26–28 August 2019.
20. Shankar, S.; Irving, L. Applying Machine Learning to Product Classification. 2011. Available online: https://cs229.stanford.edu/proj2011/LinShankar-Applying%20Machine\%20Learning%20to%20Product%20Categorization.pdf (accessed on 10 August 2022).
21. Haynes, C.; Palomino, M.A.; Stuart, L.; Viira, D.; Hannon, F.; Crossingham, G.; Tantam, K. Automatic Classification of National Health Service Feedback. *Mathematics* **2022**, *10*, 983. [CrossRef]
22. Ghahroodi, R.Z.; Ranji, H.; Rezaei, A. Using Machine Learning Classification Algorithms in Official Statistics. *J. Stat. Sci.* **2021**, *15*, 119–146. [CrossRef]
23. Gweon, H.; Schonlau, M.; Kaczmirek, L.; Blohm, M.; Steiner, S. Three Methods for Occupation Coding Based on Statistical Learning. *J. Off. Stat.* **2017**, *33*, 101—122. [CrossRef]
24. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2018**, arxiv:1810.04805.
25. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735—1780. [CrossRef]
26. Shen, Y.; Wang, X.; Tan, Z.; Xu, G.; Xie, P.; Huang, F.; Lu, W.; Zhuang, Y. Parallel Instance Query Network for Named Entity Recognition, In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, Dublin, Ireland, 22–27 May 2022.
27. Fei, L.J.; Liu, H.; Wu, J.; Zhang, S.; Teng, M.; Ji, C.; Li, F. Unified Named Entity Recognition as Word-Word Relation Classification. *Proc. AAAI Conf. Artif. Intell.* **2022**, *36*, 10965–10973.
28. Spark, J.K. A statistical interpretation of term specificity and its application in retrieval. *J. Doc.* **1972**, *28*, 11–21. [CrossRef]
29. Rajaraman, A.; Ullman, J. *Data Mining. Mining of Massive Datasets*; Cambridge University Press: Cambridge, UK, 2011; pp. 1–17.
30. Mikolov, T.; Chen K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. *arXiv* **2013**, arxiv:1301.3781.
31. Joulin, A.; Grave, E.; Bojanovski, P.; Mikolov, T. Bag of Tricks for Efficient Text Classification. *arXiv* **2016**, arxiv:1607.01759.
32. Pennington, J.; Socher, R.; Manning, C. GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1523–1543.
33. Saraswat, M. superml: Build Machine Learning Models Like Using Python's Scikit-Learn Library in R. R Package Version 0.5.3. 2020. Available online: url=https://CRAN.R-project.org/package=superml (accessed on 10 August 2022).
34. Wijffels, J. word2vec: Distributed Representations of Words. R Package Version 0.3.4. 2021. Available online: https://CRAN.R-project.org/package=word2vec (accessed on 10 August 2022).
35. Mouselimis, L. fastText: Efficient Learning of Word Representations and Sentence Classification using R. R Package Version 1.0.1. 2021. Available online: https://CRAN.R-projet.org/package=fastText (accessed on 10 August 2022).

36. Selivanov, D.; Bickel, M.; Wang, Q. text2vec: Modern Text Mining Framework for R. R package version 0.6. 2020. Available online: https://CRAN.R-project.org/package=text2vec (accessed on 10 August 2022).
37. Mertler, C.; Vannatta, R. *Advanced and Multivariate Statistical Methods*, 2nd ed.; Pyrczak Publishing: Los Angeles, CA, USA, 2002.
38. Ooi, H. glmnetUtils: Utilities for 'Glmnet'. R package version 1.1.8. 2021. Available online: https://CRAN.R-project.org/package=glmnetUtils (accessed on 10 August 2022).
39. Xu, S. Bayesian Naïve Bayes classifiers to text classification. *J. Inf. Sci.* **2018**, *44*, 48–59. [CrossRef]
40. Majka, M. naivebayes: High Performance Implementation of the naïve Bayes Algorithm in R. R Package Version 0.9.7. 2019. Available online: https://CRAN.R-project.org/package=naivebayes (accessed on 10 August 2022).
41. Wu, X.; Kumar, V.; Quinlan, J.R.; Grosch, J.; Yang, Q.; Motoda, H. Top 10 algorithms in data mining. *Knowl. Inf. Syst.* **2008**, *14*, 1–37. [CrossRef]
42. Therneau, T. Atkinson, B. rpart: Recursive Partitioning and Regression Trees. R Package Version 4.1-15. 2019. Available online: https://CRAN.R-project.org/package=rpart (accessed on 10 August 2022).
43. Kotsiani, S.B.; Tsekouras, G.E.; Pintelas, P.E. Bagging Model Tress for classification Problems. In *Advances in Informatics. PCI 2005*; Bozanis, P., Houstis, E.N., Eds.; Springer: Berlin/Heildeberg, Germany, 2005; Volume 3746.
44. Meyer, D.; Dimitriadou, E.; Hornik, K.; Weingessel, A.; Leisch, F. e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. R Package Version 1.7-9. 2021. Available online: https://CRAN.R-project.org/package=e1071 (accessed on 10 August 2022).
45. Kuhn, M. caret: Classification and Regression Training. R Package Version 6.0-91. 2022. Available online: https://CRAN.R-project.org/package=caret (accessed on 10 August 2022).
46. Quinlan, J. *C4.5: Programs for Machine Learning*, 1st ed.; Elsevier: Amsterdam, The Netherlands, 2014.
47. Hornik, K.; Buchta, C.; Zeileis, A. Open-Source Machine Learning: R Meets Weka. *Comput. Stat.* **2009**, *24*, 225–232. [CrossRef]
48. Kuhn, M.; Johnson, K. *Applied Predictive Modeling*; Springer: Berlin/Heidelberg, Germany, 2018.
49. Kuhn, M.; Quinlan, R. C50: C5.0 Decision Trees and Rule-Based Models. R Package Version 0.1.6. 2022. Available online: https://CRAN.R-project.org/package=C50 (accessed on 10 August 2022).
50. Breiman, L. random forest. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
51. Wright, N.M.; Ziegler, A. ranger: A Fast Implementation of random forest for High Dimensional Data in C++ and R. *J. Stat. Softw.* **2017**, *77*, 1–17. [CrossRef]
52. Cortes, C.; Vapnik, V. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
53. Haykin, S. *Neural Networks and Learning Machines*; Pearson Education: New York, NY, USA, 2009.
54. Venables, W.N.; Ripley, B.D. *Modern Applied Statistics with S*, 4th ed.; Springer: New York, NY, USA, 2002.
55. Cover, T.M.; Hart, P.E. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory* **1967**, *13*, 21—27. [CrossRef]
56. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; ACM: New York, NY, USA, 2016.
57. Chen, T.; He, T.; Benesty, M.; Khotilovich, V.; Tang, Y.; Cho, H.; Chen, K.; Mitchell, R.; Cano, I.; Zhou, T.; et al. xgboost: Extreme Gradient Boosting. R Package Version 1.5.2.1. 2022. Available online: https://CRAN.R-project.org/package=xgboost (accessed on 10 August 2022).
58. Van der Maaten, L.J.P.; Hinton, G.E. Visualizing Data Using t-SNE. *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.
59. Krijthe, J.H. Rtsne: T-Distributed Stochastic Neighbor Embedding using a Barnes-Hut Implementation. 2015. Available online: https://github.com/jkrijthe/Rtsne (accessed on 10 August 2022).
60. Hand, D.J; Till, R.J. A simple generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Mach. Learn.* **2001**, *45*, 171–186. [CrossRef]
61. Gardini, M.; Bagli, E.; Visani, G. Metrics for Multi-Class Classification: An Overview. *arXiv*, **2020**, arxiv:2008.05756.