

# Semi-supervised machine learning with word embedding for classification in price statistics

Hazel Martindale, Edward Rowland, Tanya Flower

---

## 1. Introduction

Research into alternative data source for consumer price statistics is a key recommendation in the Johnson Review into UK consumer price statistics [1]. Currently, the ONS produces the Consumer Prices Index including owner occupiers' housing costs (CPIH) monthly from a basket of goods and services. Every month, the price of products in the basket must be manually collected through a combination of sources, including price collectors visiting stores, collecting prices from websites and catalogues, and by phone.

Alternative data sources have the potential to improve the quality of these consumer price statistics through increased coverage and more timely data. There are two sources of alternative data we are interested in – web scraped data and point of sale transaction data. Both cover a much wider range of products and in much larger quantities than is possible with manual price collection. For example, we have recently started collecting web scraped data for a number of items including clothing [2]. In November 2018, the web scraped data contained around 900,000 price quotes for clothing items. This compares to approximately 26,000 price quotes from the manual collection, a sample size increase of nearly 35 times.

However, while the manually collected data is much smaller in volume, each individual price quote is carefully checked and categorised into ONS's item classifications [3], an extension of COICOP5 [4], by expert price collectors. The accuracy of the data is therefore very high. Due to the sheer volume of web scraped or scanner data that will be collected, it is impossible to manually classify every single item. Hence, we need to develop automated methods to classify the data to ONS item categories. These should be as accurate as possible to mitigate any reduction in classification accuracy while benefiting from the increased product coverage of the far larger data sets we expect from alternative data sources.

The sophistication of the methods required for automated classification varies greatly. For some items this is relatively simple as there are well defined rules that can be easily applied. For example, the current manual collection definition for laptops states that this category should not include refurbished products. A simple filter can remove these items by filtering on the word refurbished in the product name. However, other items, such as clothing, have complex collection definitions that rely on human judgement, i.e. what constitutes a fashion top?

To replicate these judgements a more complex machine learning (ML) approach using features created from the text descriptions of items is required. We have a defined classification structure that lends itself to a supervised ML method. A supervised classifier attempts to learn rules to divide the data into provided categories (in this case, the ONS item categories). However, in order to learn how to perform the splitting, the algorithms need data that is already labelled into the desired categories. In most instances, we have only very limited labelled data points and it requires significant resource to create more. This gives us two challenges; how do we use text information in an ML classifier and how do we increase the number of labels so that we can train a supervised classifier without the need to manually label huge amounts of data?

This paper proposes an innovative new approach to these challenges using web scraped clothing data as a pilot study. We apply methods similar to those used previously in [5] to create numerical representations of text descriptions. We then make use of the semi-supervised machine learning techniques ‘label propagation’ and ‘label spreading’ to expand a small, representative number of labels to the rest of the dataset to train, test and evaluate a number of supervised classifiers.

## 2. Data

We have chosen to begin with web scraped clothing data as it presents a particularly hard challenge. For most clothing categories, the ONS collection definitions are either ambiguous or hard to apply given the data available. For example, deciding what constitutes a “fashion top” is subjective and requires human judgement. The collectors sometimes use features that are not included in the web scraped data to establish if a product belongs to a category, such as if a shirt opens fully at the front, or what the sleeve length or neckline is. These uncertainties in the clothing category definitions make clothing hard to classify and methods developed here should apply to many other items in the basket.

While we are waiting for a sufficient time series to build up from the mySupermarket collection, we are using a web scraped dataset from WGSN (World’s Global Style Network), a global trend authority specialising in fashion. They collect daily prices and other information from a number of fashion retailers’ websites. When this was supplied in 2015, WGSN obtained data for 37 clothing product categories and 38 retailers in the UK, including a mixture of high street and online only retailers. This has subsequently increased. The web scraped data includes price, product and retailer information. Women’s clothing data were provided for the period September 2013 to October 2015 and the men’s clothing data were provided for the period August 2014 to October 2015.

The dataset is complex containing around 70 ONS categories in one dataset. This includes many items used in the existing CPIH basket as well as some clothing that would not be included, such as girl’s dresses or women’s socks, and some non-clothing items retailers chose to include in the clothing sections such as gift cards, perfume or shoes.

**Table 1: Categories used in the clothing data with the number of each category present in the sample.**

Item Description	Number of Unique products
'Junk' Items	981
Women's Coat (SEASONAL)	430
Men's Casual Shirt, long or short sleeved	379
Women's Sportswear Shorts	65
Women's Swimwear	593
Boy's Jeans (5-15 Years)	512
Girl's Fashion Top (12-13 years)	411
Men's Pants/Boxer Shorts	235
Men's Socks	239

This dataset therefore represents a multiclass classification problem where a product can belong to one of several categories. In this development phase, we use a small number out of all possible clothing categories to test the method (**Erro! Fonte de referência não encontrada.**). This includes a junk category which contains a range of products that fall outside of the chosen categories. For example, perfume, vases, toys and any clothing items not falling into one of the nine categories. The nine categories have been chosen as a small amount of labelled data was already available for these categories to inform the classification process [6], [7].

Using these categories, we construct two datasets to test and develop this approach. Each contain 3485 entries to mimic two months' worth of data. This effectively provides a training and parameter optimisation set and a second month as the out-of-sample test set to evaluate the method's generalisability.

### 3. Methods

Web-scraped data, such as the WGSN dataset, tends to combine data from several websites for the same products. As each website has a different layout, structure and information about products, there are differences in the features and completeness from product to product. For example, some features are unavailable from certain retailers. As the website information is intended for a human audience many of the attributes associated with a product are long text product names and descriptions. These require significant pre-processing before they can be used in the classification process.

Due to the complex and diverse nature of the web-scraped data we are required to employ several different methods in combination to expand the labelled dataset, create features and perform classification. Below we will describe the methods used, these fall in to four major categories: Natural language processing (NLP), increasing the amount of labelled data, classification methods and assessment metrics. Within each of these categories we will discuss the methods appropriate for our use case.

### 3.1 Word Embeddings

Creating numerical features from text data is a complex task with many different algorithms having been developed. These range from basic word counts to complex neural network-based methods that attempt to capture context.

#### 3.1.1 Count vectorisation

Count vectorisation is one of the simplest ways to encode text information into a numerical representation first introduced by [8]. Here the number of times every word in the total corpus is found in a given sentence or phrase is used as the vector.

For example, if we take the phrases:

Sentence1: "There is a black cat",

Sentence2: "A black cat and a black ball",

Sentence3: "Is there a black cat?"

**Table 2: Demonstration of simple word vectorization method, count vectorization.**

	Black	there	is	Cat	And	Ball	A
Sentence1	1	1	1	1	0	0	1
Sentence2	2	0	0	1	1	1	2
Sentence3	1	1	1	1	0	0	1

This results in the vectors shown in **Erro! Fonte de referência não encontrada..** These vectors have encoded the information contained in the sentences, however this method offers no context for the words and different sentences can have the same representation as there is no encoding of word order. The above example demonstrates this as the first and last sentence have the same words (and therefore the same vectors) but different meaning.

One way to try and mitigate this is to use several words together, these are known as bi-grams for two words and tri-grams for three words. This leads to much larger vectors where pairs of words are considered in addition to single words. If we considered the bigrams for the example in Table 2, we would no longer have the same vectors as "There is" and "is there" are different bigrams. Therefore, we can distinguish between two sentences containing the same words but with a different word order. In this work we make use of bi-grams in addition to single words when calculating the word vectors.

This simple word vectorisation method gives a base line method for word vectorisation. They are conceptually simple to build. However, there is a significant drawback when applying this a count vectorisation model to new sentences. If the new sentences contain words not previously seen when training a classifier using features derived from count vectorisation, these new words cannot be used as features in the classifier.

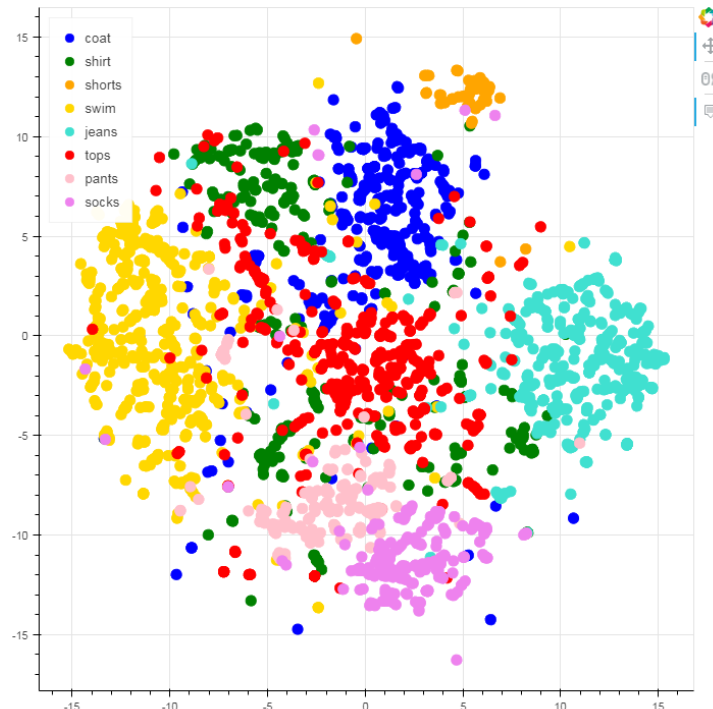
### 3.1.2 TF-IDF

Term Frequency – Inverse document frequency (TF-IDF) builds on the count vectorisation method. It is a more sophisticated method of building the vectors as it attempts to give more prominence to more important words. TF-IDF does this by introducing a weighting to the words based on how commonly they appear in the total document. The logic of this is that the most frequently-used words such as *the* and *a* will occur very often in a sentence but they are not important for distinguishing the meaning of text in a document. Those less common but potentially more important words will be upweighted by the inverse document frequency defined below. In this work we use the Scikit-Learn implementation of the TF-IDF [9]. In this implementation the term frequency is taken to be the count vectorization described in the previous section.

The IDF is given by:

$$idf(t) = \log \frac{n_d}{df(d, t)} + 1$$

where the  $n_d$  is the total number of documents/sentences and  $df(d, t)$ .

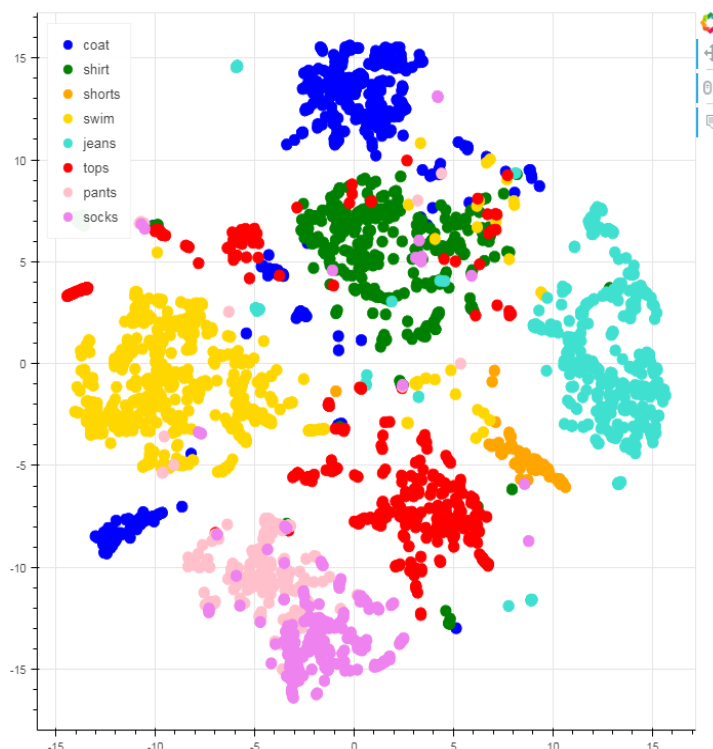


**Figure 1: Two-dimensional projection of the word vector for a sample of clothing items generated using the TF-IDF algorithm.**

Like the count vectorizer however it still does not account for the context of the words and cannot deal with unfamiliar words that have not appeared before in previous documents. As with the count vectorizer, by extending the vectors to using bi or tri grams in addition to the single words it is possible to allow for some context for the words, but this is limited. In this work we make use of bi-grams in addition to the single words when training the TF-IDF vectors. In **Erro! Fonte de referência não encontrada.** we show a two-dimensional projection of the TF-IDF vectors for our dataset. These vectors have been projected using t-SNE algorithm that is well suited to visualising high dimensionality data in two or three dimensions [10]. We see in Figure 1 that some clothing items are better separated with greater clustering in vector space than others. Shorts (orange) and jeans (cyan) are reasonably well separated. However, shirts (dark green) and top (red) show much more overlap which makes sense given many of the same words would be used to describe these clothing items.

### 3.1.3 word2vec

A more complex way to create numerical features from text is the word2vec algorithm [11]. This method makes use of a shallow two-layer neural network to convert the words to word embeddings in a high dimensional vector space.



**Figure 2: Two-dimensional projection of the word vector for a sample of clothing items generated using the word2vec algorithm.**

Word2vec works by creating a model to predict the next word given a target word and the surrounding context words of the rest of the document. For example, for the sentence “The mouse eats cheese from the box” if the target word was “eats” the model would be attempting to predict “cheese” given the rest of the words in the sentence. The weights of the final hidden layer in this neural network are then taken as the word vectors to represent words of all the input sentences also known as the training corpus. A text corpus is a collection of structured written texts.

The resulting word vectors for words which are similar will have similar vectors and so will occupy a similar area of the vector space in which they have been embedded. This means that unlike the count vectorization and the TF-IDF method the context in which words are found is taken into consideration when generating the vectors. This is because the neural network was trained to predict the next word given the surrounding words so must consider context.

For this application the words that we wish to embed are the product names, usually short phrases of typically 4-8 words. To obtain a single word vector for the product we average the word vectors, obtained from each word in our sentence (e.g. a product name) together. This provides a single averaged vector for each product. The resulting word2vec vectors for our clothing dataset are shown in **Erro! Fonte de referência não encontrada..** The word vectors here are showing several distinct clusters for the different clothing items with less overlap than was seen for the TF-IDF vectors.

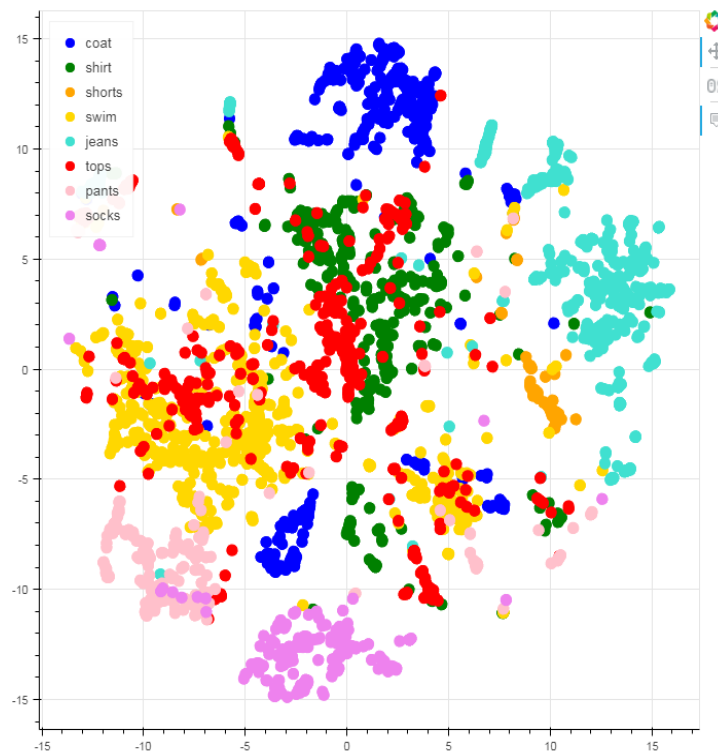
### 3.1.4 fastText

fastText is an algorithm that is built on very similar principles to word2vec [12]. One of the major differences between the two algorithms is that the fastText algorithm uses n-grams as well as whole words when constructing the predictive model. N-grams are substrings of a word of length N. For example, for the word “apple”, the bi-grams are ap, pp, pl, le.

By using the n-grams in addition to the whole words, the resulting word embeddings are better for rare words and are also able to cope with out-of-corpus words. This gives an advantage over the other word embedding techniques described previously as they are not able to handle out-of-corpus words. If an out-of-corpus word is encountered but the n-grams that make it up are available in the training data, a vector is constructed out of the n-gram vectors.

A disadvantage of the fastText and word2vec methods are that they are complex to train. This could be mitigated by using pre-trained vectors, but these would not be specialised for the use case of our data. For both the word2vec and fastText vectors we make use of the gensim python library [13] to train our word vector models on the clothing product names. The resulting fastText vectors for the clothing dataset are shown in **Erro! Fonte de referência não encontrada..** In this projection, as with word2vec, the vectors show significant clustering of different clothing types in different areas of vector space. The vectors here do not appear to be separated as in the word2vec version but as they are projected into two dimensions it is hard to say for certain that one vectorisation is better.





**Figure 3: Two-dimensional projection of the word vector for a sample of clothing items generated using the fastText algorithm.**

## 3.2 Label propagation for increasing the amount of labelled data

Supervised machine learning methods require a large amount of accurately labelled data to train the algorithms to ensure that the classifiers are generalisable and able to better cope with new products. However, obtaining data that has already been labelled to the required ONS item level categorisation of the CPIH basket is not possible and the task of manually labelling a large amount of data is onerous.

Getting a limited number of labelled data points is often a more feasible option. It is then possible to expand the amount of labelled points available using the methods described below to generate a larger set of labels for training.

### 3.2.1 Fuzzy matching

We start the labelling process with a small number of manually produced labels, typically between 10 and 20 per ONS item category in our data. We use string matching methods to provide a first pass method to increase the number of products that have labels. These methods will assign labels to only those products that have a similar product name to those original labels.

The first of these is the Levenshtein distance. This counts the number of changes (inserting, changing or deleting a character) required to turn one string into another. The second is the partial ratio method of the fuzzy-wuzzy python library. This method is better at comparing strings of different lengths. For instance, *blue stripy jumper* and *stripy jumper* would be an exact match for the partial ratio whereas they would have a Levenshtein distance of five as Levenshtein requires five-character insertions to match the strings “*blue*”.



The third metric we use is the Jaccard distance. This takes all the characters in both strings as two sets and calculates the distance as follows:

$$1 - \frac{\text{length of the intersection of both strings represented as sets}}{\text{length of the union of both strings represented as sets}}$$

For the jumper example, the Jaccard distance is 0.15<sup>1</sup>. A perfect match would be 0 as the intersection and union of the strings would have the same length.

Each of these metrics measure slightly different things and therefore have different advantages. It is hard to say which metric would give a definitive match. The Levenshtein distance is not well suited to strings of different length as each difference in length will be counted as a change even if the substrings match. For our use case it is likely that the string lengths will vary quite substantially and so the Levenshtein distance will provide conservative matches.

By contrast, the partial ratio technique will create matches for any product with the same substrings. This may provide more matches, but these will generally be of lower quality as just because the substring is an exact match the surrounding context words may mean it is not in reality a match.

The Jaccard distance also does not deal well with strings of different length as the extra length will dominate the length of the union of the strings and so different length strings will be classed as dissimilar. This again will provide conservative matches. This method does though have a potential advantage in that it does not preserve character order, so a black hat and hat black would be a match. The sets used to calculate the intersection and union do not consider character order. This is desirable behaviour for some strings and not for others.

Due to these trade-offs between the metrics we use all three metrics together and only accept a match if two out of the three distance metrics agree.

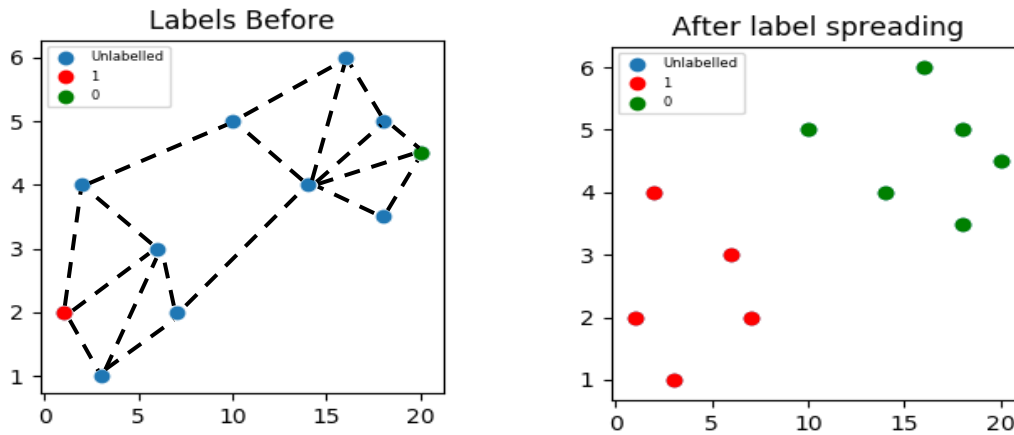
As this approach will be used as the seed labels for the next step, this is a conservative method of finding matches as we want to minimise the false positives. In effect, what this approach does is find unlabelled products with very similar product names to the products in our labelled dataset and assigns the unlabelled product that same label.

### 3.2.2 Label propagation

Label propagation is part of a group of semi-supervised machine learning algorithms. Given our limited pool of labelled data points, semi-supervised learning can use these known labels in the learning process to steer an unsupervised learning approach to label the rest of the data, hence semi-supervised.

In this work we make use of the label propagation and label spreading algorithms from the Scikit-Learn python library [9]. These algorithms were introduced by [14] and [15] and work in very similar ways. Both are graph (or network) based methods and make the essential assumption that data points that are close in the graph to a labelled point will likely share the same label.

<sup>1</sup> When the strings are considered as set only unique character are preserved. The resulting intersection of the two strings as sets is 11 and the union is 13. The resulting Jaccard distance is  $1 - 11/13 = 0.15$ .



**Figure 4: Illustration of the principles of the label propagation algorithm. The left-hand panel shows before the propagation algorithm, including the seed labels and the graph. The right-hand panel shows the result of running the label propagation algorithm correctly labelling the two clusters.**

In **Erro! Fonte de referência não encontrada**, we illustrate what is meant by a graph in this context. The graph is a collection of points or nodes in some  $n$ -dimensional space that are connected via edges or links. These edges can either connect all of nodes to all other nodes or can only connect some. The left-hand panel of **Erro! Fonte de referência não encontrada** shows a two-dimensional example graph where each node is connected to its 3 nearest neighbours.

The graphs are not limited to two dimensions and can be constructed across the entire  $n$ -dimensional feature space. The graph is critical to the label propagation process and is most commonly produced using the  $K$  Nearest Neighbour (KNN) algorithm. The principle of the KNN algorithm is to join each point to its  $K$  nearest neighbours where  $k$  is an integer value. In the example in **Erro! Fonte de referência não encontrada**, we have created the graph with a value of  $k=3$ . If  $k$  is equal to the number of points, this would be described as a fully connected graph.

The process of creating the graph is computationally intensive as each point must calculate the distance to all other points in order to establish which are the  $k$  nearest. This will limit the use of this algorithm in extremely large datasets. In this work we use Euclidean distance when calculating the graph, however other metrics maybe applied instead.

Once the graph is created the label propagation or spreading is then performed to “push” labelled points through the graph and so grow the labelled dataset. This is shown in **Erro! Fonte de referência não encontrada**. In the left-hand panel the graph and the seed labels are shown and in the right-hand panel the result of the label propagation algorithm is shown. This shows that the red label has been propagated to the cluster nearest to it in the graph and similarly for the green.

The label propagation algorithm can be described in the following steps:

First a probabilistic transition matrix  $T$  is defined, this is the distance between points weighted so that close nodes are given more weight and further away nodes less and is given by:

$$T_{ij} = P(j \rightarrow i) = \frac{w_{ij}}{\sum_{k=1}^{l+u} w_{kj}}$$

Where  $T_{ij}$  is the probability of a label jumping from node  $i$  to  $j$  and  $w_{ij}$  is the distance between the node  $i$  and  $j$  normalized by the sum of the distance between all points. The other component of the initial set up is the labels of all points. For the labelled points these will be defined already and the unlabelled points are assigned a dummy label of -1. This vector of labels is given as  $Y$ .

The new labels of  $Y$  are then given by

$$Y \leftarrow TY$$

The  $Y$  vectors are then renormalized, and the final step is to “clamp” the original labels back to be the same as the initial labelled points. This ensures that these known labels do not get changed during the label propagation process. It can be shown that this algorithm will iterate to convergence. For details see [15].

### 3.2.3 Label spreading

The other algorithm used in this work is label spreading. The basic idea of this algorithm is the same as that outlined above for label propagation except that the initial labels are not firmly clamped. Instead some proportion of the labels are allowed to change. This is particularly helpful if the input labels are likely to be noisy and contain inaccurate examples, which is the case for our data as we have applied the fuzzy matching step to expand our initial labelled set. Here the new labels are given by:

$$\tilde{Y}^{(t+1)} = \alpha \mathcal{L} \tilde{Y}^{(t)} + (1 - \alpha) Y^{(0)}$$

Where  $t$  denotes the iteration,  $Y^{(0)}$  indicates the initial labels, and  $\alpha$  is a number between 0 and 1 and  $\mathcal{L}$  is the normalised graph Laplacian see [15]. This enables a proportion of the initial labels to change. Similar to the label propagation method in 3.2.2, it can be proven that this algorithm does converge. For further details on the mathematics and graph theory behind this algorithm see [14], [16] and [17].

We use the label spreading variant of label propagation here as the initial labels are noisy and not well defined for this dataset. This is because we have applied the first fuzzy matching step that expands the number of labelled products but introduces noise into the labels.

As with the fuzzy matching (section 3.2.1) we have performed the label propagation 3 times. In each of these times we use a different one of the word vectorisation methods TF-IDF, word2vec and fastText. Then as with the fuzzy matching we only accept a label if two of the three label propagation results agree.

The newly labelled datasets can then be used to train a classifier, whether that is a machine learning method or simple filtering.

## 3.3 Classification method

There are many different algorithms that have been developed to perform classification tasks. Not all algorithms are suitable for all datasets or problems and have varying levels of complexity and transparency. Below we discuss the methods that are considered in this work.

### 3.3.1 Decision tree

A decision tree is an easy to interpret form of machine learning classifier. It can conceptually be thought of a flowchart. The tree is built up of branches and nodes. At each node, a decision rule will split the data in two and this will then continue to the next node and the next. This process of splitting continues until a leaf is reached. Each leaf node is the target category being classified.

The decision tree is a supervised machine learning technique. During a training stage the algorithm determines which of the possible splits on the data will give the purest leaf nodes. In this work we use the Scikit-Learn version of the decision tree algorithm with the Gini impurity for selecting the decision rules [9]. For further details of the algorithm see [18] or [19].

Decision trees have the significant advantage that they are easy to interpret compared to many other machine learning metrics. As they are conceptually similar to a flow chart it is possible to see and understand why the algorithm has made its splitting choices. Another major advantage of this algorithm is that they can cope with a range of input data, both continuous and categorical. They also do not require the data to be scaled to the same numeric ranges. This means that they are quicker to develop as they do not require such large amounts of feature engineering. The final advantage that is relevant for our use case is that they are non-parametric and so do not make assumptions about the distribution of the data. As our data is not linearly separable this makes the decision tree a useful algorithm for our work.

However, they also have some disadvantages. It is possible for the decision trees to become very large and complex reducing the interpretability of the tree. They also have the potential to suffer from overfitting and a lack of generalisability. The tree is developed to give very good results for the training data. If the tree is allowed to perform many splits on the data, it will be very accurate on the training data but too overly specialised to generalise to new data.

### 3.3.2 Random Forest

The random forest is an ensemble model built on decision trees. Instead of training only one tree, many trees are trained, and the classification is the mode of the output for all trees. Again, we use the Scikit-Learn implementation of the random forest algorithm.

To build a random forest a sample of the training data is taken and a decision tree built, and then another sample, with replacement, is taken and another tree built. This is continued until the desired number of trees are built. In addition to a random set of the data, a random set of the features are also used. Once the trees in the forest are built, the data is then run through each tree and the modal category is taken as the final classification for each observation.

The random forest has an advantage over the decision tree in that it is better at generalizing to unseen data, less prone to overfitting and generally perform well. However, they come at the cost of reduced interpretability as they are the average result of many trees.

### 3.3.3 Support vector machines

The final classifiers considered in this work are support vector machines (SVM). These classifiers work by finding the hyperplane(s) that divides the data into the required categories with the largest margin between the hyperplane and the data. See [18] for further detail about the mathematics of this method.

SVM's can work with both linear and non-linearly separable datasets. By employing the kernel trick the non-linear data can be transformed into linear. Then the algorithm can find the correct hyperplane. The SVM algorithm is known to generalize well to unseen data making them good predictive classifiers. They do not however provide probability estimates unless they are used in conjunction with cross-validation (section 3.4). One drawback of the SVM is that it is not suited to very large datasets particularly when using a non-linear kernel. Again see [18] for further details.

### 3.4 Assessment metrics

There is a myriad of ways to assess a classifier and often one single metric is not sufficient to capture the quality. In most situations and certainly in our datasets we do not have a balanced dataset with equal amounts of data in each category. With our data, the junk category is much larger than the others. This can lead to the overall performance appearing better than for some of the individual items when looking at the assessment metrics.

For our classification task, accurately identifying the smaller classes is more important than the junk class. This means that we must use an assessment metric that accounts for this class imbalance. We use F1-score in a one-versus rest comparison for each label to assess the quality of the classifier for each item in our data. This helps get around the unbalanced data problem as F1-score is good at assessing classifier quality for low prevalence data where the negative class (all other labels) is larger than the positive class (the label we are assessing our classifier performance on).

The F1-score gives an assessment of classifier performance for each class. We can take the performance of each class given by its F1-score and take a mean, weighted by inverse prevalence to give an overall weighted Macro F1-score. We compute class prevalence to use as a weight ( $W_x$ ) as a ratio of the number of products in a class ( $x$ ) and the overall size of the data by the following:

$$W_x = \frac{n_x}{n_{all}}$$

We then compute the weighted Macro F1-score for  $c$  classes using the below:

$$F1_{macro} = \sum_{x=1}^c \frac{1}{W_x} F1_x$$

It is necessary to understand the input data and examine the outputs across all categories, so we cannot take the Macro F1-score alone. Instead we have to examine the performance of the classifier across each class individually. For example, a classifier could have a very strong Macro F1-score but perform poorly on a small number of classes meaning it may not be a good solution for our purposes.

It is also important to understand why the classification is being performed and the impact on the desired result of a misclassification. For our use case of price statistics, the classification is necessary to keep the price distribution to a true representation of the given category. If misclassified objects have a significant impact on the price distribution, then that is an error that can't be tolerated. However, if the impact is negligible then the error is more acceptable. There is a trade-off between accuracy and development time. It is most desirable to develop the simplest method that provides the desired level of accuracy for the problem in hand. We have not assessed this aspect of classification as this is beyond the scope of this paper, but this is an avenue for future development.

#### 4. A multi-step process for classifying clothing data

Using the clothing data, we apply the methods described in section 3 in a multi-step process. In summary, we start with 10 manually labelled products for each category that we will use as seed labels to propagate and label the dataset.

In **Erro! Fonte de referência não encontrada.**, we show this process in a flow diagram. In stage 1, the three fuzzy matching methods are applied to generate a slightly larger sample of labels. In stage 2, we only accept a match if two of the three methods agree. This is because none of the metrics are perfect and we attempt to reduce the potential for introducing unwanted bias from using only one metric. We also clean the created labels to remove any unrealistic labels. For example, if the category is Boy's Jeans then any items with Men in the gender column will be removed, or items with product location dress would also be removed. This is a conservative way of cleaning and relies on the gender or location column being accurate, which may not be true in all cases.

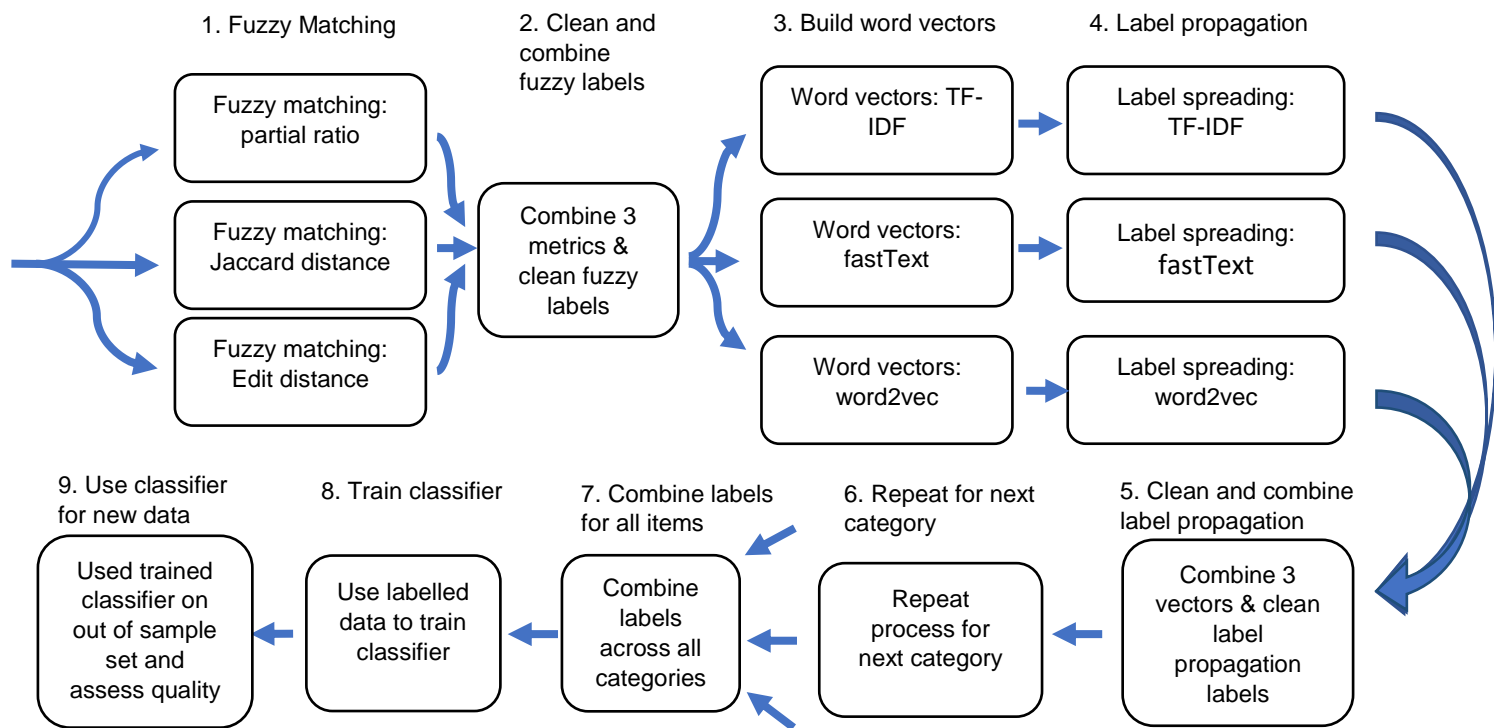
In stage 3 and 4, we use word embedding and label spreading in combination to propagate the labels through the dataset for each of the three methods (TF-IDF, fastText and word2vec). Again, in stage 5 we only accept a label if two of the three methods agree. We also clean the data again to remove any unrealistic labels. For the labelling, we are prioritising precision of the labelling over recall as the labels being correct is the most important thing at this stage. In the label propagation algorithm, a probability level for the assigned label is provided. In our pipeline we require this to be over 70% before a label is accepted in two of the three kinds of word vectors. This is to ensure cleaner labels, anything that is not confidently junk or one of the given nine categories is given a value of -1 or unlabelled. We have found that our process gives better results if we perform the labelling as several one-vs-rest pipelines. This means for our dataset with nine target categories we run the pipeline from fuzzy matching (stage 1) to label propagation (stage 5) nine times before combining the resulting labels in stage 7. If a product has been assigned more than one label by the label propagation algorithm, e.g. it has been labelled as women's shorts and women's swimwear, it would be assigned a new label of -1, unlabelled. This is because it is not possible to know which of the two categories are correct and so the label is not reliable.

These newly created labels can then be used to train a classifier, stage 8 in Figure 5. The process is as follows - use the labelled data to train the classifier, apply the classifier to the next month's data (the out of sample set), and then assess the accuracy of the classifier based on the results from out of sample set.



In training the classifiers, we use the word vectors as features as well as the category, retailer, age and gender columns. The latter three columns contain categorical variables that must be encoded in order to be used by the versions of the classification algorithms. To do this we currently use label encoding. This is where all the possible categories are assigned a number e.g. if the categories were tree, plant, flower they would be assigned 0, 1, 2. All future items would be then encoded with one of the numbers according to their category. Using these additional categorical features in addition to the word vectors will help the classifier distinguish adults and children's clothing. When training the classifier, we choose to use only one of the three type of word vectors with the categorical variables.

In the following section we assess the results of this process at the different stages as well as the effect of changing the word vectors used in training the classifier.



**Figure 5: Diagram showing the full pipeline for labelling and classifying the data for one item. This diagram shows all the steps the data moves through including those where the data divides for multiple methods or metrics. This entire process will be run for each category from fuzzy matching (stage 1) to the end of label propagation (stage 5) and then the final labels combined and used in a classifier.**



## 5. Results

We use two datasets for assessing the results of this method. One is passed through the labelling process and then used to build the classifiers, the training dataset. The other is held back for testing how well the developed classifiers generalise to unseen data, the 'out of sample' dataset. This second dataset is only used in the final test of classifier accuracy. Both datasets have 3845 data points. After label propagation, although all of the training data is labelled, only those items that have been confidently assigned a category label are taken forward to train the classification algorithms.

### 5.1 Fuzzy Matching

The first step to increase the number of labels is fuzzy matching, as shown in **Erro! Fonte de referência não encontrada.** We perform the matching for each item in turn using the 10 manually labelled products as the seed labels.

**Erro! Fonte de referência não encontrada.** shows the category by category results, the number of labels assigned and the precision (a measure of false positives). As the results of the fuzzy matching will later be used in the label propagation algorithm we want high levels of precision. Any false positives will be exaggerated in the propagation. We see in general very high levels of precision for all the items. The women's sportswear shorts find the most labels through fuzzy matching, this is likely due to small size of this category.

**Table 3: Results of the fuzzy matching part of the label propagation pipeline across all 9 target categories.**

	Number of labels assigned	Precision	Percentage of true items labelled
Women's Coats	188	0.99	47.6%
Men's Casual Shirts	248	0.98	65.6%
Women's Sportswear shorts	61	0.89	93.8%
Women's Swimwear	314	1.0	56.3%
Boys Jeans	403	0.97	78.8%
Girl's fashion top	150	0.91	43.1%
Men's Underwear	154	1.0	71.2%
Men's socks	174	0.96	73.1%

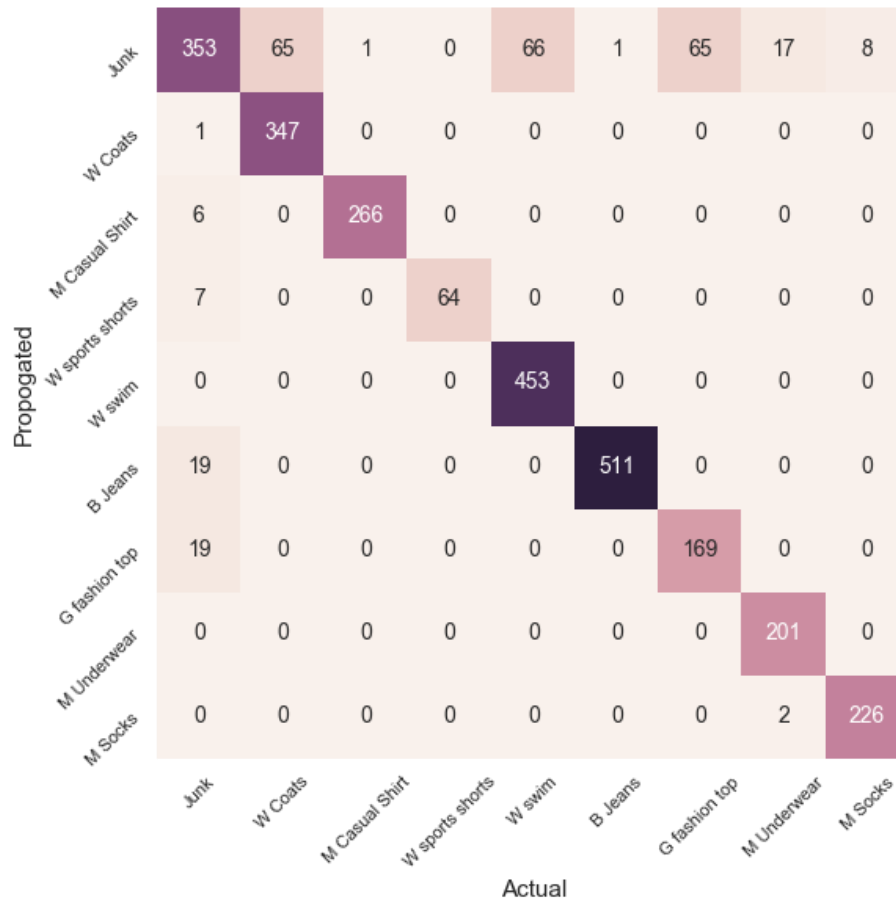
After the fuzzy matching stage, we find we have expanded the 91 original labels to 1692 labels, this equates to 48.5% of the total data passed through the fuzzy matching method. We have chosen the fraction for accepting a match for each of the three metrics in order to maximise the precision. This requires further tuning to see if there are further improvements that can be made to the quantity of labels without compromising the quality.

## 5.2 Label propagation

In order to assess how well the label propagation is doing at predicting the labels we compare the assigned label to the actual labels in our data. It is working well in this sample dataset, further expanding the number of labelled data points to 2802 or 73% of the total dataset. In **Erro! Fonte de referência não encontrada.** we show the classification metrics for label propagation across all categories and the average metric weighted by size of each category.

**Table 4: Classification metrics for the label propagation algorithm used on the clothing data sample**

Item	Prevalence in dataset	% labelled by label propagation	Precision	Recall	F1-score
Junk	981	58.7	0.61	0.87	0.72
Women's coats	430	82.1	1.00	0.85	0.92
Men's Casual shirt	379	71.8	0.98	1.00	0.99
Women's sportswear shorts	65	109.2	0.90	1.00	0.95
Women's swimwear	593	75.7	1.00	0.86	0.93
Boy's Jeans	512	103.3	0.96	1.00	0.98
Girls Fashion top	411	45.5	0.90	0.72	0.80
Men's Underwear	235	86.0	1.00	0.91	0.96
Men's Socks	239	97.0	0.98	0.98	0.98
Weighted macro Average			0.92	0.90	0.91



**Figure 6: Confusion matrix for the label propagation algorithm. The categories are those shown in the table, where W stands for women’s, M for Men’s, G for girls and B for Boys.**

Figure 6 shows the confusion matrix resulting from the label propagation. The confusion matrix shows how items have been classified both correctly and incorrectly. The top left to bottom right diagonal show the correctly classified items while other elements show miss classifications. For example, with girl’s fashion top 169 items have been correctly classified, 19 junk items have wrongly classified as a girl’s top and 65 actual tops have been classified wrongly as junk. The quality of the labelling varies across the categories, some performing very well while others require improvement. There are two main reasons for the discrepancies. First, the item is not well represented in the training data. This is the case for the women’s sports wear shorts as there are approximately 60 labelled products compared to the 400-500 for other categories.

The other problem categories are those with subjective ONS item definitions such as fashion tops or the very broad junk category. In the case of junk items and fashion tops, it is likely that the initial labelled set is not fully covering the items that could be included in the category, so some are missed. The other possibility is that because the junk category in particular is so broad, some clothing items can be easily miss-classified into it. The categories included in the dataset (Table 1) cover nine out of 67 categories at ONS item level. It is possible that items in the junk category are very similar to those nine categories that have been specified. For example, a girl's coat item would be classified as a junk item, but it is also a close match to women's coats with very similar names. Therefore, these items would occupy very similar positions in our feature space causing these to potentially be misclassified. For both the fashion items and the junk category this could be improved with more input labels for the label propagation and increasing the coverage for ONS items in the initial labels.

### 5.3 Using the labels in a classifier

The aim of this work is to develop a classification framework that is able to deal with complex data as well as simpler structures. Now we have a method for creating large and accurately labelled data we can now develop a classifier. As previously stated, we use the labels resulting from the label propagation pipeline to train the classifier and a second dataset to assess the classifiers quality. In using these two datasets as fake months we aim to mimic the way the true web-scraped CPIH data is collected.

As the clothing data requires several columns, such as product name, age and gender to make a decision about how a product should be categorised we choose to apply a machine learning method. For this work we have chosen to focus on decision trees, random forests and non-linear support vector machines as the clothing data is not linearly separable. We compare the three methods below. In training the classifiers we do not combine the three different word vectorisation methods as we did for label propagation, instead we compare the results of using each in turn as part of the set of features with the categorical variables.

Table 5, 6 and 7 show the results from the three classification methods, using vectors from TF-IDF, fastText, and word2vec respectively. We see that all of the classifiers are performing reasonably with assessment metrics over 80% across all methods. We also show that the labels automatically generated through label propagation are of a sufficiently high quality to be able to train a classifier. All of the classifiers are also indicating that they are slightly overfitting in training as there is a drop in the performance when going from training to test. We are reluctant to draw too many conclusions about this as the samples used in this work are not a full set of all ONS item level products in the data. Even if labels were available for all ONS clothing items, ONS item level categories are not complete so there are clothing items that do not fall into *any* category. Nevertheless, a larger sample that is representative of the whole range of clothing at ONS item level in each category may help improve the generalisability of the classifiers.

**Table 6: Weighted macro average assessment metrics for the 3 machine learning classifiers using word vectors built using the TF-IDF method. The training data is that labelled by label propagation and the test data is manually labelled data.**

	Training data			Test data		
	Precision	Recall	F1-score	Precision	Recall	F1-score
Non-linear SVM	0.89	0.88	0.89	0.85	0.85	0.87
Decision Tree	0.88	0.87	0.87	0.85	0.81	0.82
Random Forest	0.88	0.88	0.88	0.85	0.84	0.84

**Table 5: Weighted macro average assessment metrics for the 3 machine learning classifiers using word vectors trained with the fastText model. The training data is that labelled by label propagation and the test data is manually labelled data.**

	Training data			Test data		
	Precision	Recall	F1-score	Precision	Recall	F1-score
Non-linear SVM	0.90	0.87	0.89	0.82	0.80	0.87
Decision Tree	0.86	0.85	0.86	0.82	0.81	0.80
Random Forest	0.87	0.88	0.87	0.82	0.80	0.80

**Table 7: Weighted macro average assessment metrics for the 3 machine learning classifiers using word vectors trained with the word2vect model. The training data is that labelled by label propagation and the test data is manually labelled data.**

	Training data			Test data		
	Precision	Recall	F1-score	Precision	Recall	F1-score
Non-linear SVM	0.89	0.89	0.89	0.85	0.83	0.84
Decision Tree	0.88	0.88	0.88	0.84	0.85	0.83
Random Forest	0.88	0.88	0.88	0.86	0.86	0.86

Similar to the results from label propagation, we see considerable variation when we look at the results at the category level (Appendix A, B and C has the full break down of the results across all categories). The problem categories remain similar to those we saw in section 5.2 for the label propagation. The men's casual shirt is one such category where the classifiers are failing to distinguish casual and formal. This can sometimes be a hard challenge even when manually labelling the data. We are not surprised that this nuance in the category definition is not being reproduced in the classifier.

Another area where we have high misclassifications is Women's swimwear. The ONS item level definition for this category only accepts full sets as products for inclusion in the consumer basket, therefore separates should be excluded from this category even though they are sold in shops as woman's swimwear items. However, inspecting the data for women's swimwear shows that a large amount of the misclassified products are separate bikini tops and bottoms that are not joined into a set. This means that even though the classifiers are not performing well in terms of our labels, conceptually they are doing a reasonable job. We may need to develop the method further or include more manual checks for items such as Woman's Swimwear. An alternative solution would be to change the ONS item definition to include these products in the collection definition. For the women's sportswear shorts we suspect that the problem is one of over fitting in the training data as there is only 80 examples from this class in the training data. A larger sample of this class in training would likely improve the results from this category.

Another way to improve the method is to use active learning (described in section 6). Alternatively, for the case where we do not currently have the required features (e.g. to identify bikini tops and bottoms or formal shirts), we can create more features. For example, searching the product name for the word bottom would allow us to create a feature to help in distinguishing the bikini types. The process of feature creation is not easily automated and requires an expert understanding of both the data and category definitions.

## 6. Considerations using this in production

When taking this work further into a system that will be run every month with new data there are several things that need to be taken into consideration. Most of these are around maintaining the quality of the classification through time, changing website structure and product churn.

In the case of product churn, it is likely that the number of unclassifiable items will small. These could be identified by having a very low probability of being assigned a given category by the classifier. These items would then be manually labelled by humans. These newly labelled data will then be added to the old labelled data and the classifier re-trained. This is a form of machine learning called active learning and the specific version described here is uncertainty sampling [20]. In this form of learning the classifiers are continually updated and evolve as new data is available, which also reduces the need for a complete retrain periodically.

If the chosen assessment metrics continue to fall below the acceptable threshold, even with using active learning, a complete retrain may be required. This would be a significant effort where the whole process of label propagation and then training the classifier would need to be rerun with new seed labels. This should not be required very frequently but rather when a significant change is made to either the categories being collected or the data source structure (for example, a website changes the way it displays product names).

In order to establish if a retrain is required, or likely to be required soon, a continual monitoring of the classification metrics is required. To perform the monitoring, it will be necessary to label a sample of items from each category manually every month. This will require a level of human interaction with the process on a monthly basis. Further work is required to explore the ways to draw the sample for labelling to give the best estimate of total classifier quality.

## 7. Future Work

The pipeline developed here has shown significant potential as a method to create larger labelled datasets when only a limited pool of labels is available. There are several ways that this work can be extended and improved.

The first of these is to look at methods for generating better coverage of junk labels. The pipeline suggested in this paper does generate junk labels through those that are very dissimilar by fuzzy matching or label propagation to any seed labels. But these junk labels are limited in their coverage of the vast and ever-changing potential junk items that could be scraped from a website. This is a particular issue around events like Christmas or Easter, where retailers can stock seasonal items in every website category. If we were able to have a better coverage of the junk items, the final classifiers generalisability may improve. In order to achieve this, we may need to look at new learning strategies such as Positive-Unlabelled learning that do not require such a well-defined junk class, see [18].

Another consideration is how well this method will scale to very large datasets that may be used for price statistics in the future. The graph creation required for the label propagation is a very computationally expensive algorithm which is not easily parallelised. This leads to a significant bottleneck for using this algorithm on many millions of points. We will investigate the possible ways to mitigate this. Currently the most likely scenario is to use sample of the data rather than the whole dataset. How to generate truly representative samples to train the classifier on without prior knowledge of the data set will require further research.

A further consideration is tuning the hyper-parameters in each of the learning steps. This is something that has not been rigorously explored in this current work but has the potential to improve the performance of the labelling and the classifiers. Training the word2vec and fastText models are one example where there are many hyper-parameters that are hard to tune. This problem may be alleviated by using pretrained vectors made available by the creators of these algorithms, for example we could use pre-trained models trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset or on Common Crawl. These datasets contain billions of input words used in training. These may also help improve the performance of the algorithms.



## 8. Conclusions

This work has presented a method for labelling and classifying complex web scraped data for use the consumer price statistics. We have developed a pipeline through which it is possible to generate a large enough labelled dataset with which to train a traditional classifier using a very limited set of manually labelled products. The use of the label propagation algorithm gives a way to generate large labelled datasets without requiring many man hours of manual labelling. The resulting labels have been shown to be accurate in our test example dataset with approximately 70% of the data successfully labelled to greater than 90% precision. This is promising for expanding the work to other web scraped items from the CPIH basket. The method developed here is also not only applicable to price statistics but can help reduce the labelling burden for other classification problems.

## 9. References

- [1] P. Johnson, "UK Consumer Price Statistics: A Review," UK Statistics Authority, 2015. [Online]. Available: <https://www.statisticsauthority.gov.uk/reports-and-correspondence/reviews/uk-consumer-price-statistics-a-review/>. [Accessed 2019].
- [2] Office for National Statistics, "New web data will add millions of fast-moving online prices to inflation statistics," 9 July 2018. [Online]. Available: <https://www.ons.gov.uk/news/news/newwebdatawilladdmillionsoffastmovingonlinepricestoinflationstatistics>.
- [3] Office for National Statistics, "Consumer price inflation basket of goods and services: 2019," 11 March 2019. [Online]. Available: <https://www.ons.gov.uk/economy/inflationandpriceindices/articles/ukconsumerpriceinflationbasketofgoodsandservices/2019>.
- [4] United Nations Statistics Division, "COICOP Revision," United Nations Statistics Division, 2018. [Online]. Available: [https://unstats.un.org/unsd/class/revisions/coicop\\_revision.asp](https://unstats.un.org/unsd/class/revisions/coicop_revision.asp).
- [5] Office for National Statistics, "ONS working paper series number 14 – Unsupervised document clustering with cluster topic identification," 19 April 2018. [Online]. Available: <https://www.ons.gov.uk/methodology/methodologicalpublications/generalmethodology/onsworkingpaperseries/onsworkingpaperseriesnumber14unsuperviseddocumentclusteringwithclustertopicidentification>.
- [6] Office for National Statistics, "Research indices using web scraped price data: clothing data," 31 July 2017. [Online]. Available: <https://www.ons.gov.uk/economy/inflationandpriceindices/articles/researchindicesusingwebscrapedpricedata/clothingdata>.
- [7] Office for National Statistics, "Analysis of product turnover in web scraped clothing data, and its impact on methods for compiling price indices," 2017. [Online]. Available:

<https://www.ons.gov.uk/methodology/methodologicalpublications/generalmethodology/currentmethodologyarticles>.

- [8] K. S. Jones, *A statistical interpretation of term specificity and its application in retrieval*, 1972.
- [9] F. Pedregosa FABIANPEDREGOSA, V. Michel, O. Grisel OLIVIERGRISEL, M. Blondel, P. Prettenhofer, R. Weiss, J. Vanderplas, D. Cournapeau, F. Pedregosa, G. Varoquaux, A. Gramfort, B. Thirion, O. Grisel, V. Dubourg, A. Passos, M. Brucher, M. Perrot andÉdouardand, a. Duchesnay and F. Duchesnay EDOUARDDUCHESNAY, “Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot,” 2011.
- [10] L. Van Der Maaten and G. Hinton, “Visualizing Data using t-SNE,” 2008.
- [11] T. Mikolov, K. Chen, G. Corrado and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” 2013.
- [12] P. Bojanowski, E. Grave, A. Joulin and T. Mikolov, “Enriching Word Vectors with Subword Information”.
- [13] R. Řeh ůrek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, 2010.
- [14] X. Zhu and Z. Ghahramani, “Learning from Labeled and Unlabeled Data with Label Propagation,” p. , 1 2003.
- [15] D. Zhou, O. Bousquet, T. N. Lal, J. Weston and B. Schölkopf, “Learning with Local and Global Consistency”.
- [16] G. Bonaccorso, *Mastering Machine Learning Algorithms*, Packt Publishing, 2018.
- [17] G. Bonaccorso, *Machine Learning Algorithms - Second Edition*, Second ed., Packt Publishing, 2018.
- [18] C. C. Aggarwal, *Data Classification*, Chapman and Hall/CRC, 2015.
- [19] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, 2 ed., Springer-Verlag New York, 2009.
- [20] D. D. Lewis and J. Catlett, “Heterogeneous Uncertainty Sampling for Supervised Learning,” in *Machine Learning Proceedings 1994*, 2014.

## Appendix A: Classification with TF-IDF detailed results

The detailed breakdown of the results of the classifiers trained with the TF-IDF vectors. These are the results when applying the trained model to the test data. Figure 7 presents the confusion matrices for all three classifiers followed by the performance metrics in **Erro! Fonte de referência não encontrada.** These show the problem categories where the classifiers are not performing as well as for other categories such as girls fashion top, women’s swimwear and men’s casual shirt.



Figure 7: Confusion matrices for the three classification methods with TF-IDF vectors. Top left panel is the decision tree, top right random forest and bottom is the non-linear SVM. These are the confusion matrices for the test dataset.

**Table 8: Category by category breakdown of the metrics for the classifiers using TF-IDF vectors. As with the confusion matrices this is the result using the test dataset.**

	Decision Tree			Random Forest			Non-linear SVM		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-score
Junk	0.70	0.68	0.69	0.71	0.62	0.66	0.69	0.87	0.77
Women's Coats	1.00	0.83	0.91	1.00	0.86	0.92	0.99	0.89	0.94
Men's Casual shirts	0.97	0.69	0.81	0.69	1.00	0.82	0.96	0.71	0.82
Women's Sportswear shorts	0.62	1.00	0.77	0.41	1.00	0.58	0.78	1.00	0.88
Women's Swimwear	0.99	0.89	0.94	1.00	0.89	0.94	0.99	0.91	0.95
Boys Jeans	0.96	1.00	0.98	0.96	1.00	0.98	0.96	1.00	0.98
Girls Fashion top	0.45	0.70	0.55	0.78	0.77	0.77	0.86	0.63	0.73
Men's Underwear	0.98	0.90	0.94	0.97	0.93	0.95	0.91	0.93	0.92
Men's Socks	0.98	0.98	0.98	0.99	0.99	0.99	0.92	0.97	0.95

## Appendix B: Classification with fastText detailed results

The detailed breakdown of the results of the classifiers trained with the fastText vectors. These are the results when applying the trained model to the test data. Below we present the confusion matrices for all three classifiers followed by the performance metrics in **Erro! Fonte de referência não encontrada.** These show the problem categories where the classifiers are not performing as well as for other categories for example men’s casual shirts and women’s sportswear shorts.

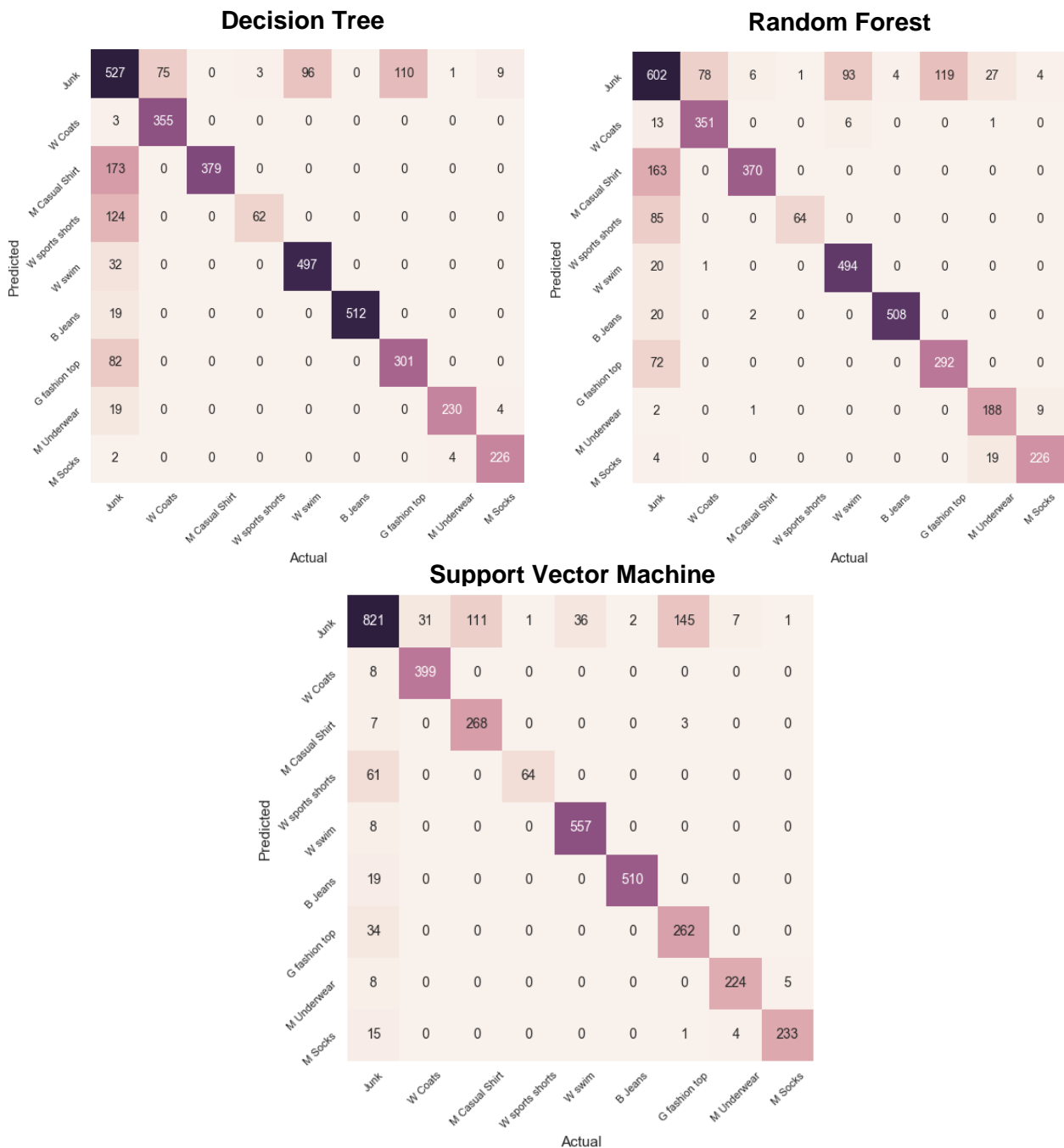


Figure 8: Confusion matrices for the three classification methods with fastText vectors. Top left panel is the decision tree, top right random forest and bottom is the non-linear SVM. These are the confusion matrices for the test dataset.

**Table 9: Category by category breakdown of the metrics for the classifiers using fastText vectors. As with the confusion matrices this is the result using the test dataset.**

	Decision Tree			Random Forest			Non-linear SVM		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-score
Junk	0.64	0.54	0.58	0.64	0.61	0.63	0.69	0.88	0.77
Women's Coats	0.99	0.83	0.90	0.95	0.82	0.88	0.99	0.87	0.93
Men's Casual shirts	0.69	1.00	0.81	0.69	0.98	0.81	0.96	0.71	0.82
Women's Sportswear shorts	0.33	0.95	0.49	0.43	0.98	0.60	0.78	1.00	0.88
Women's Swimwear	0.94	0.84	0.89	0.96	0.83	0.94	0.99	0.91	0.95
Boys Jeans	0.96	1.00	0.98	0.96	0.99	0.89	0.96	1.00	0.98
Girls Fashion top	0.79	0.73	0.76	0.80	0.71	0.75	0.88	0.62	0.73
Men's Underwear	0.91	0.98	0.94	0.94	0.80	0.86	0.91	0.91	0.91
Men's Socks	0.97	0.95	0.96	0.91	0.95	0.93	0.92	0.97	0.95

## Appendix C: Classification with word2vec detailed results

The detailed breakdown of the results of the classifiers trained with the word2vec vectors. These are the results when applying the trained model to the test data. Below we present the confusion matrices for all three classifiers followed by the performance metrics in **Erro! Fonte de referência não encontrada.** These show the problem categories where the classifiers are not performing as well as for other categories such as men’s casual shirt and girls fashion top.

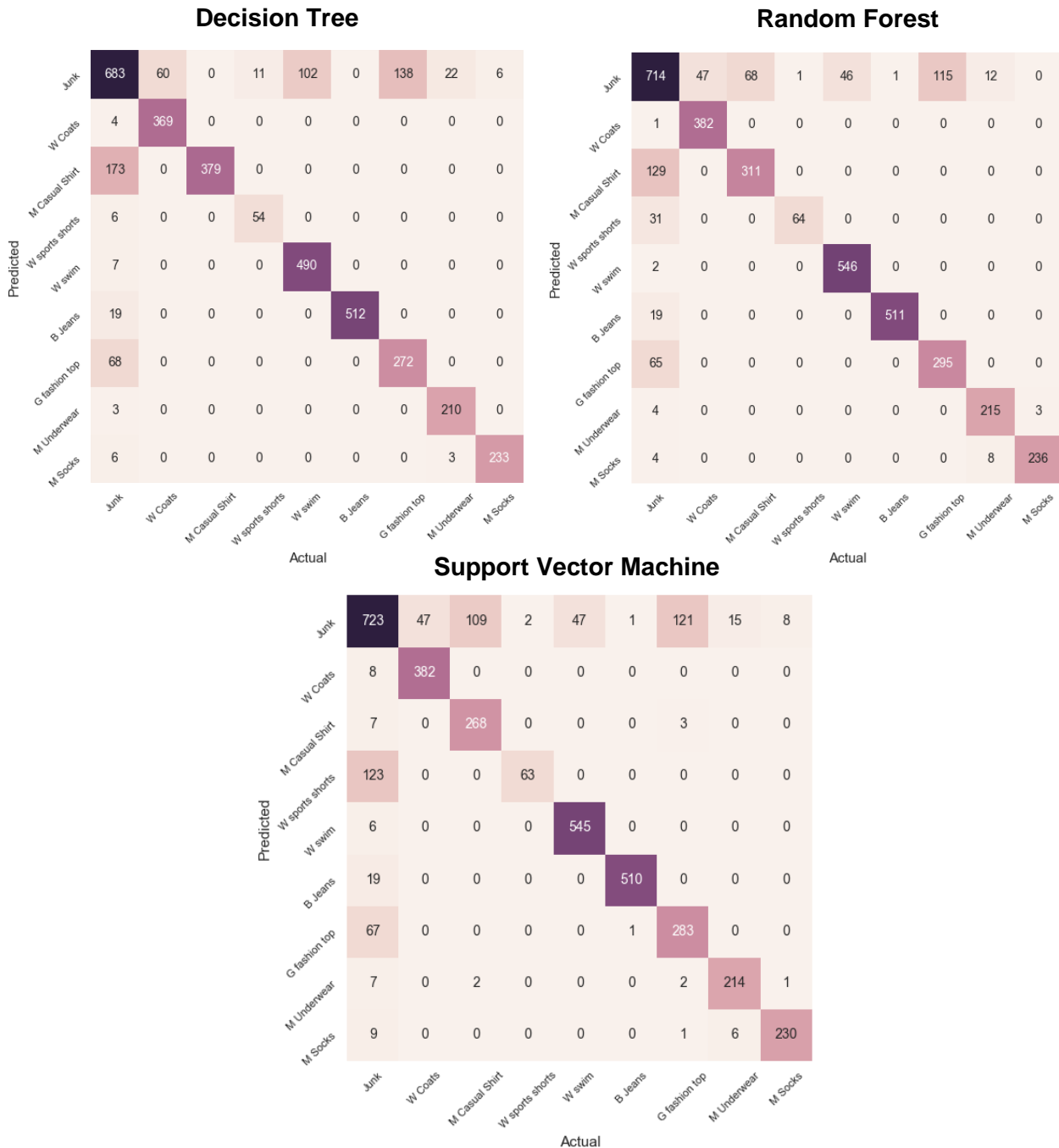


Figure 9: Confusion matrices for the three classification methods with word2vec vectors. Top left panel is the decision tree, top right random forest and bottom is the non-linear SVM. These are the confusion matrices for the test dataset.



**Table 10: Category by category breakdown of the metrics for the classifiers using word2vec vectors. As with the confusion matrices this is the result using the test dataset.**

	Decision Tree			Random Forest			Non-linear SVM		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-score
Junk	0.67	0.70	0.69	0.71	0.74	0.72	0.67	0.75	0.71
Women's Coats	0.99	0.86	0.92	1.00	0.89	0.94	0.98	0.89	0.93
Men's Casual shirts	0.69	1.00	0.81	0.71	0.82	0.76	0.96	0.71	0.82
Women's Sportswear shorts	0.90	0.83	0.86	0.67	0.98	0.80	0.34	0.97	0.50
Women's Swimwear	0.99	0.83	0.90	1.00	0.92	0.96	0.99	0.92	0.95
Boys Jeans	0.96	1.00	0.98	0.96	1.00	0.98	0.96	1.00	0.98
Girls Fashion top	0.80	0.66	0.73	0.82	0.72	0.77	0.81	0.69	0.74
Men's Underwear	0.99	0.89	0.94	0.97	0.91	0.94	0.95	0.91	0.93
Men's Socks	0.96	0.97	0.97	0.95	0.99	0.97	0.93	0.96	0.95