NLP Deceitful Review Detection on e-Commerce and Social Media

Laura Débora Fernández Basquero,

Juan Pablo Guerrero Ortiz,

Alejandra Lloret Calvo,

Ainhoa Rodrigo Bolea,

Cristián Fernando Rodríguez,

Marta Roig Atienza

Contents

NLP Deceitful Review Detection on e-Commerce and Social Media

Abstract. The world of misinformation and fake reviews are gaining ground on online shopping, negatively affecting both consumers and retail e-commerce business. The Big Data techniques, such as Natural Language Processing (NLP), Machine Learning (ML) and Deep Learning (DL), have currently a leading role on spotting the falsehoods on internet comments. These tools have contributed to ensure the veracity and quality of reviews, helping companies to maintain user's confidence and fidelity. Due to the wide range of possibilities that Big Data offers and the novelty of the field, different NLP techniques have been applied along this study, in order to train and compare alternative classification models. For this purpose, an Amazon reviews data-set based on different product categories has been used. Our findings suggest that the Word Embedding NLP technique along with the classical logistic regression is the best fitted model and fake reviews detection would improve by training several models per product category.

Keywords: Natural Language Processing, Machine Learning, Deep Learning, fake reviews, tokenization, embedding

Media Summary

Nowadays, online shopping has reached its peak due to the development of platforms such as Amazon and it has become even stronger in the midst of this pandemic (COVID-19). The e-commerce's key of being at the forefront of retail is simple: instant access to a wide range of products in a single click for everyone everywhere. Conversely, not all that glitters is gold: fake reviews pose a potential threat to consumer confidence.

In this sense, consumers have been surrounded by a murky world of fake product reviews and it is getting harder to spot them. Besides that, many e-commerce vendors learned the hard way that fake reviews can damage their reputation, killing the consumer's trust. Despite lagging consumer confidence in online reviews, the world of big data brings a glimmer of enlightenment to the recovery of consumer's faith.

Natural Language Processing (NLP) has allowed the interactions between computer and human language. This is the key to process online reviews in order to classified them as fake or true. To do this, Machine Learning (ML) and Deep Learning (DL) models are the perfect tool to accomplish that goal.

Taking these ideas together is what has led the development of this study. Different NLP techniques, ML and DL models have been explored to determine which of them works better on detecting fake reviews. Moreover, as Amazon is the biggest e-commerce platform and it is in the spotlight of fake reviews, it has been used an Amazon product reviews data-set.

The research has been split into two different parts. The first part is related to Theoretical Framework (Section 1), where it is briefly introduced the different big data technologies applied along the study: Natural Language Processing, Machine Learning and Deep Learning models. The second part, Practical Framework (Section 2) presents the main findings. Finally, Section 3 concludes and set out the further research in this field.

1. Theoretical Framework

1.1. NATURAL LANGUAGE PROCESSING FRAMEWORK.

Natural Language Processing (NLP) is a branch of artificial intelligence that allows computers to learn in an accurately way to process language components such as words, sentences or paragraphs in order to get a correct understanding of spoken or written language.

1.1.1. *Cleaning data.*

As in any data analysis, data cleaning is a key preliminary step to achieve better modelling and predictive power. In NLP framework, it acquires even more importance due to we are working with text, something for which currently computers are not prepared without a previous treatment. There are many steps involved in this process, such as converting upper case letters into lower case, removing punctuation marks, deleting line breaks, among others.

1.1.2. *Lemmatization.*

Lemmatization is a technique used in NLP to convert the words in a text into its base form, taking into consideration its context. [1] In this way, it is possible to capture more relationships between words with the same meaning/sense in spite of not being written in their base form.

1.1.3. *Tokenization.*

Tokenization is the previous step to convert each word into a numeric value. By this process it is possible to separate words from text into entities called tokens.

1.1.4. *Vectorization and embedding techniques.*

Since **Machine Learning models** could not understand letters as an input, it turns out to be mandatory to find a way to transforming the vectors of words into vectors of numbers. Therefore, this paperwork is implementing several techniques to accomplish our goal: to transform each of the reviews given by the users or clients of the website into a feature composed by numbers in each observation of the data-set. According to each solution implemented to get this aim, the results showed up that model accuracy varies onto the numbers representing those words.

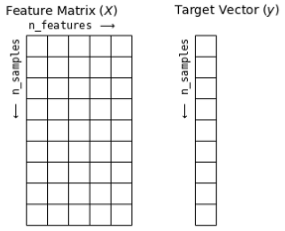
This process is denominated as **Vectorization Techniques** due to the need to get an object of numbers, the state of art on NLP techniques to solve the problem propose general solutions and advance methods. But, before to explain in what it is necessary to get the vectorize text observation, let us introduce core concepts to this process:

[1] Some lemmatization techniques also identify the 'part-of-speech' (POS) tag for the word in its specific context and extract the appropriate lemma.

(1) Document: Each text in each observation of the data-set. Normally, an object of text sequence. E.g. A tweet, book, sentence, so on.
(2) Corpus: A collection of documents. It is used to training the unsupervised models and extract topics from new documents in new observations. E.g. In our data the corpus length is of 21.000 documents. See EDA for more information.
(3) Vector: In this case, it is a mathematical convenient representation of the document. In this case, we have shown before the pre-steps undertaken to get a vector representation, such as eliminate the "stop–words", [2] Lemmatization, Eliminate punctuation marks and transform all words to lowercase and getting on with the tokenization, which Tokenization breaks up the documents into words. By definition, a feature vector could be defined as 'an n-dimensional vector of numerical features that represent some object', in this case TEXT.
(4) Model: Algorithm for transforming vectors from one representation to another.

In general terms, to transform a word into an integer is a problematic decision, so it is necessary to denote a feature related to the corpus. Hence, we are summarising how frequent are the words inside the whole corpus as it could go into the Machine Learning models to predict the dependent variable and classify the entire line text as 'Fake' or 'Real'. Once, it transforms the words in integers we grouped these on feature vectors inside a matrix or build an object like a dictionary to get the number of all the words on each document.

In case of the matrix, we would have n-columns with feature vectors as words in each document and n-rows as n-observations has the data-set. As illustrated in the next figure:



In summary, machine learning models are using the counting of the occurrences of any word and their associations with the rest of words in the train data to discriminate between the categories of the dependent variable, or even calculate their impact when each of these both categories have occurred on real and learn from this fact.

Mainly, we have different methods to vectorizing the tokens to achieve different methodologies to count the occurrences of the word or also to counting the relation of the words with the rest of words in the corpus and transform these relations in numbers.

We could split those methods up to two groups:

[2]Thanks to the Natural Language Tool Kit (NLTK) in Pyhon, it has been possible to ignore the stop words found within the data-set. This tool kit has a predefined list of stop words in different languages, including English, which has been used in this research. For farther information please visit: https://www.nltk.org/

(1) **Counting Methods** In general terms, the next three different kind of vectorization techniques are in essence doing the same but with slightly modifications to counting the number of times that each word occurs in the text.

Their main differences are the way to count the occurrence of each word and the output matrix generated by each of them. Those methods are the following:

(a) Count Vectorization
(b) N-Grams
(c) Term Frequency – Inverse Document Frequency (TF-IDF)

(2) **Embedding Word Methods:** Basically, these methods are using simple neural networks to get a pre-trained space vector or embedding space, where it is used to map all the tokens respect to the rest of words in the same document.

(a) Word2Vec
(b) *GloVe*

1.2. DEEP LEARNING FRAMEWORK.

1.2.1. *Deep Neural Networks.*

A **Convolutional Neural Network** (CNN) uses hidden layers which are called convolutional layers. These layers have parameters that are learned so they filter inputs to extract the most useful information. Those nets use pooling layers too and other types of layers are also used in a Deep Learning Model, each of them transforms the weighted input received using different functions.

CNN is one of the main categories to do images recognition and classifications. The input of an image is a two-dimensional matrix of numbers and the convolutional layers consists of multiple filters that are able to detect edges, corners and other kinds of textures. However, we are working with sequential data. So, instead of image pixels, the input are sentences represented as a matrix and we will work with one-dimensional convolutions. Nevertheless, the idea stays the same, you pick patterns in the sequence which become more complex with each added convolutional layer. Each row of the matrix corresponds to one token, it represents a word in this case. That is how CNN works for (NLP).

1.2.2. *Word Embeddings.*

The method that we have used to deal with representing words as vectors is **Word Embeddings**. It represents words as word vectors (word embeddings) which are trained to map syntactic meaning into geometric space (embedding space) using an embedding matrix that has been already trained (*GloVe, 300 dimensions*).

It is necessary to configure the learning process. We want to use the binary cross entropy and the Adam optimiser. Due to the fact that CNN is an iterative process, it is necessary to specify the number of iterations (**epochs**) and how many samples we want to use in one epoch (**batch size**). We have chosen 7 epochs and 100 batch size because it achieves the best accuracy and it does not over-fit the model (the loss of the validation data does not start rising again).

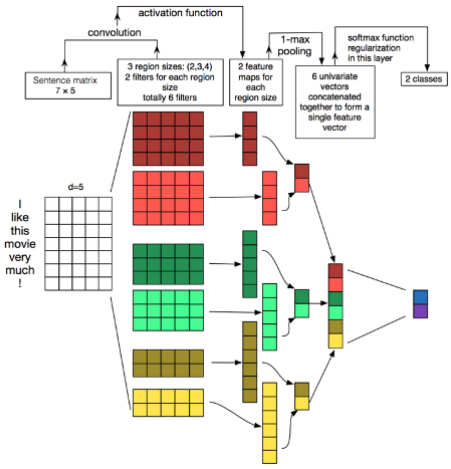


Illustration of a CNN architecture for sentence classification. Here we depict three filter region sizes, each of which has 2 filters. Every filter performs convolution on the sentence matrix and generates feature maps. Then 1-max pooling is performed over each map. Thus a uni-variate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states.

1.3. MACHINE LEARNING FRAMEWORK.

There are several text data columns in our dataset such as *"Product_ Category"*, *"Review_ Tittle"*, or the most important feature *"REVIEW_ TEXT"*, which is the most extensive text data in our dataset. In order to feed up the models in a proper way, any of these texts has to be represented into numbers. Therefore, this paperwork is implementing several techniques to accomplish our goal: to transform each of the reviews given by the users or clients of the website into a feature composed by numbers in each observation of the dataset. According to each solution implemented to get this aim, the results showed up that model accuracy varies onto the numbers representing those words.

Then, each vectorization technique is explained with the dataset used.

(1.a) Counting Methods: Count Vectorization

Count Vectorization method creates a document-term matrix where the entry of each cell will be a count of the number of times that word occurred in that document. It assigns an index to the word and count its occurrence in each document. Hence, this matrix has a number of columns equals the number of unique words and a number of rows equals as the number of documents.

E.g. In our dataset we have 35.002 unique words within 21.000 documents. Storing 735.042.000 of elements between 0 or 1 (see Table 1).

(1.b) Counting Methods: N-Grams

This method creates a document-term matrix where columns instead of representing unique words, represent all combinations of adjacent words of length n in the corpus. The adjacent length could be from a pair, trio, quartet or more words combinations. E.g. Bigram, Trigram, Four-gram, and so on.

In this case, the method is picking up the occurrence of the "n-gram" length selected in each document and returns a matrix with the number of rows as the number of documents and number of columns as number of adjacency words possible to get in the documents. The dimension of the output matrix would be larger if the lenght of the "N-grams" used increase. The main advantage of this process is to catch up these causal and meaning relations and vectorize them, rather than only count the words frequency. Its drawback is to be costly to compute for the ML algorithms.

E.g. In our dataset, we have 436.651 combinations of 2-grams within the 21.000 documents. Storing 735.042.000 of elements between 0 or 1 (see Table 2).

(1.c) Term Frequency – Inverse Document Frequency (TF-IDF)

This method evaluates the importance of the words in each document respect to the importance of that word in the text Corpus. It counts the number of times a word appears in the document, and the inverse the document frequency of the word across the whole corpus. It helps to adjust the value of those common words appears more frequently in general, like prepositions, if, what, etc.

The aim to rank low those common words by offsetting their value with the inverse of the word frequency in the corpus is to get the more real relevancy of each word in the feature observation (or document). This more accurate measure could help ML algorithms to determine better the impact of some words with the dependent variable according to word informative relevance. In other words, a less informative word is that word present in all documents.

To calculate the value given to the word, we could use the following formula: *for a term t of a document d in a document set D is:*

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

where,

$$tf(t, d) = log(1 + freq(t, d))$$

$$idf(t, D) = log\left(\frac{N}{count(d \in D : t \in d}\right)$$

TF(t) = log (Number of times term t appears in a document d) / (length of the document or Total number of terms in the document)

IDF(t) = log(Total number of documents N / Number of documents with term t in it)

Note: In order to avoid zero division it could be a included a constant of '1' in both parts of the formula of *idf(t,D)* calculation. It would be interpreted as if an extra document was seen containing every term in the collection exactly once.

E.g. In our dataset we have 35.002 unique words within 21.000 documents. Storing 735.042.000 of elements with the resultant value of the previous formula for each word (see Table 3).

To make these matrices more readable and lighter to model computation process, we apply the concept of a 'Sparse Matrix', which allows to get only the cells of the matrix where a value different from 0 is found. See Jupyter Notebooks annexed.

(2.a) Embedding Word Methods: Word2Vec

Word2Vec was developed at Google by Tomas Mikolov, et al (2013). This word representation method provides an efficient implementation of two different architectures which are built to use the word vectorization of Continuous Bag-of-Words (CBOW) and Skip-gram methods. Those learning algorithms are based on Neural Networks, which are composed by a shallow neural network, it consists in only one hidden layer between input and the output layers. Once, one of both learning algorithms is trained, it is able to produce a word vector as output.

What Word2Vec does as an embedding method is representing the words in a vector space representation, as once before it has constructed a vocabulary from the training text data. This representation placement is done by locating the words according to their linguistic context, hence the learning algorithm analyse every document to determine the words surrounding by a specific word or backwards. The way of the semantic analysis depends on the architecture chosen between CBOW or Skip-gram. Both methods could be explained briefly, as follows:

- Continuous Bag-of-words (CBOW): It solves the shortcomings of the standard bag-of-words method, which ignores position and order of the word, and also the context relationships. On the other hand, it encodes the syntactic and semantic relations of any word calculating its features and updating them concerning neighbour or context words with the help of a back-propagation method. It means, the algorithm predicts words using the window around the word and the cosine distance between them and the word.
- Skip-grams: In this method the prediction is the words context or sequence around that word given. But the process is the same than CBOW but in a reverse order, from the look-up word to the weights of the words around.

Thus, all words are vectorized with the numbers of distance between them, due to similar words end up with a similar numerical representation.

In this research, we used to vectorized the vocabulary in our corpus from a pre-trained word vector left in the official site of Google trained on part of Google News dataset with about 3 million of words and phrases: https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit.

This pre-trained vocabulary helps out to codifying all our words related to the rest of words inside each document. In this case, the output per word is a relation vector with the rest that has to be aggregated in any form such as Mean, Maximum o Minimum value in order to ingest the model with only one feature per word in the feature vector of the observation.

The results obtained after experimented by training our own vectors to embedding the words of our corpus are really poor and there is a need to get these trained vectors.

(2.b) Embedding Word Methods: *GloVe*

GloVe was developed by Jeffrey Pennington et al (2014). in Stanford University. The core of the Global Vectorization process is represented also a vector space model but instead to only process the similarity of the words, it also uses a specific weighted log-bilinear least squares model that trains on global 'word-word' co-occurrence counting inside the corpus and thus makes efficient use of statistics.

It demonstrates to improve and outperform the results given by Word2Vec or other methods (see References [1]).

It also returns a lighter output to the model due to the matrix reduction process applied to the results that are embedding from each word. Basically, it transforms the resultant matrix using neural networks architectures to language modelling. This language modelling process is focused to building a co-occurrence matrix, which retains the information of how frequently each word occurs with only the top 10,000 most frequent words.

Despite to *GloVe* would be a mix between counting and embedding methods, generally speaking, it differs to the mentioned methods, because it reduces the dimensional matrix of vector representation, using a factorisation process to yield a lower-dimensional matrix of words and features, where each row returns a vector representation for each word in the document. But also, it combines the beneficial aspects of the Skip-gram model to make use the context information of the text to transform the word. To sum up, its main difference with Word2Vec is that *GloVe* uses the probability of co-occurrence that each word in the document related to each of present words along same

document, and it uses these probabilities to get the word vector learning. Finally, once the vector of probabilities is achieved by the model, we have to use an aggregate measure to ingest the ML models, such as Mean, Maximum or Minimum.

1.4. **H2O FRAMEWORK.**

H2O is a fully open source, distributed in-memory machine learning platform with linear scalability. H2O supports the most widely used statistical machine learning algorithms including gradient boosted machines, generalised linear models, deep learning and more. H2O also has an industry leading AutoML functionality that automatically runs through all the algorithms and their hyperparameters to produce a leaderboard of the best models. The H2O platform is used by over 18,000 organisations globally and is extremely popular in both the R and Python communities.

2. EXPERIMENTAL FRAMEWORK

2.1. **TREATMENT, ANALYSIS AND DATA DESCRIPTION.**

Since Amazon is the leading company in online shopping sector, on the Internet is available to anyone a repository of user reviews classified by product. This data repository is freely accessible and many people have extracted sub-samples to work on them. This is the case of the data-set used, as it has been downloaded from a kaggle user, where the reviews are labelled as fake or true in a balanced way at 50 % each. For more information about this repository please see: https://www.kaggle.com/lievgarcia/amazon-reviews

2.1.1. ***Preliminary Data Cleaning Analysis.***

Before analysing the data, it is necessary to apply a preliminary data cleaning in order to obtain a clearer and more orderly exploratory data analysis. The steps taken are:

- To convert the independent variable *"VERIFIED_PURCHASE"* into a binary variable.
- To replace the values of the dependent variable as: __label1__: "fake" and __label2__: "true".
- Ensuring that any variable has missing values.
- To convert all the letters into lowercase, remove text in square brackets, remove punctuation and remove words containing numbers.
- To remove stop-words.

2.1.2. ***Preliminary Feature Engineering.***

Once the preliminary data cleaning has been done, it has been created new features in order to enrich the information provided by the original data. In total has been created 4 new features:

(1) **Polarity:** Sentiment polarity which lies in the range of [-1,1] where 1 means positive sentiment and -1 means negative sentiment. It is based on `TextBlob` [3] library from Python.

[3]TextBlob: is a Python library for processing textual data. For farther information please visit: https://textblob.readthedocs.io/en/dev/

(2) **Subjectivity:** Sentiment subjectivity which lies in the range of [0,1] where 0 means objectivity and 1 means subjectivity. It is based on `TextBlob` library from Python.
(3) **Review length:** Numeric feature that contains the length of each review (measured in characters).
(4) **Word count:** Numeric feature that contains the number of words in each review.

2.1.3. ***Exploratory Data Analysis***.

Alongside this point it is exhibited the visual analysis of the data, which is one of the most important tasks in the field of text mining. Specially, it is going to be explored the content of reviews from different aspects and at different levels of details.

Also, it is going to be analysed numeric and categorical features, paying specially attention on the relations between them and their influence on the objective variable.

2.1.3.1 *Univariate Visualization Analyses*.

Firstly, it is important to know how is distributed each independent variable.

- Figure 1 exhibits the sentiment polarity of Amazon's reviews. The majority of the scores are above zero, meaning that the most of the reviews are written in a positive sense.
- Figure 2 shows the distribution of the rating set for each product by each review. Many of the rating are around 4 and 5, these results are aligned with the **Figure 1**.
- Figure 3 exhibits the distribution of the length of the reviews. The majority of the reviews could fit perfectly in a tweet (two hundred eighty characters).
- Figure 4 displays the distribution of the number of words of the reviews. In general, our population prefers to leave short reviews.
- Figure 5 shows the number of reviews of each category, as we can see it is an equidistributed dataset, which contains exactly 700 reviews of each product category.
- Figure 6 exhibits the 20 more frequent words. As it can be seen, the words highlighted are with positive meaning (see Table 4).
- Figure 7 shows the top 20 bigram and such as Figure 6, these are related with positive purchase sentiment (see Table 5).
- Figure 8 displays the top 20 trigram. Surprisingly, the top 1 is linked with a negative purchase sentiment (see Table 6).
- Figure 9 shows the top 20 Part-of-speech tagging for the reviews. The Part-of-speech is the process of classifying words into their parts of speech and labelling them accordingly. Parts of Speech are also known as word classes or lexical categories. The collection of tags used for a particular task is known as a tag set.

As follows, using the library `nltk`, we can see the description of each tag. Specifically, we are interested on the meaning of NN, JJ, CD and NNS. The vast majority are singular nouns(NN), adjectives(JJ), numerals and cardinals(CD), and plural nouns(NNS).

Firstly, we have grouped the thirty product categories in a High-Level Category classification, resulting in the following table:

High-Level Product Categories	Original Product Categories	Number of reviews
Home	Home, Home Entertainment, Home Improvement	2100
Furniture	Furniture, Lawn and Garden, Outdoors, Kitchen	2800
Electronics	Electronics, Video DVD, Video Games, Camera, Rools, PC, Wireless	4900
Music & Books	Books, Musical Instruments	1400
Health	Health & Personal Care, Beauty	1400
Baby	Baby	700
Jewellery	Jewellery, Watches, Luggage	2100
Others	Pet Products, Toys, Glocery, Office Products	2800
Sports	Sports	700
Automotive	Automotive	700
Apparel	Apparel, Shoes	1400

Previously, we have analysed the full distribution of the dataset for each relevant variable. Now, we are going to carry out some boxplot analyses for each High-Level Product Category, in order to study if there is any difference between them.

- Figure 10 exhibits the Sentiment Polarity Boxplot for each High-level product category. Some facts may be emphasised:
  - *Jewellery* and *Apparel* are the two categories with the highest median.
  - Most of the values of the distribution are accumulated above zero. Meaning that, most of the reviews are written in a positive sense.
  - *Home*, *Electronics*, *Furniture*, *Jewellery* and *Others* have many outliers in the left tail. At first glance, it seems that we are working with left heavy-tailed distributions.
- Figure 11 displays the distribution of each High-level category based on the rating. We can appreciate how all the distribution is concentrated between the ratings 4 and 5, excluding *Baby* and *Others* categories, for which the distribution is concentrated between rating 3 and 5. In short, most of the products, regardless of the category, are well rated.
- Figure 12 shows the boxplot of each high-level category based on the review length. The longest reviews are found in *Home*, *Electronics*, *Music & Books*, *Others* and *Sports* categories. The type of products that we can found among these categories normally are expensive, so the clients are grateful if them work as it is expected.

2.1.3.2 *Bivariate Visualization Analyses*.

In the following section, we are going to make use of Bivariate visualization, which consists on studying two features at a time, describing association or relationship between two features.

- Figure 13 displays the distribution of sentiment polarity of reviews based on Good (reviews with a rating of 4 or 5) or Bad rating. As it can be seen, the reviews that have higher polarity score are more likely to be well rated.
- Figure 14 shows the distribution of sentiment subjectivity of reviews based of Bad or Good rating. The distribution of good rating reviews is closer to a normal distribution, while the bad rating distribution is platykurtic, this means that the curve's tails at both sides are fatter as a result of it being higher.

- Figure 15 exhibits the distribution of rating of reviews based on if the purchase has been verified or not. There are more verified purchases than not verified and the majority of them scores with the highest rating.
- Figure 16 displays the distribution of fake and true reviews based on verified purchase or not. As we can appreciate, true (fake) reviews are more likely to be a (not) verified purchase.
- Figure 17 shows the distribution of the rating of reviews based on its length. The longest reviews are at the same time good rating reviews.

2.1.4. ***Final Data Cleaning*** & ***Data Engineering***.

2.1.4.1 *Final data cleaning*.

Aligned with the preliminary data cleaning described in the previous section, it has been necessary to add more steps in order to carry out a better treatment and modelling of the available data. These are the following:

- To clean some emojis found in the reviews using a python `emoji` [4] library.
- It has been necessary to update python stop-word list adding (I've, I'm, I) as they had not been taken into account.
- It has been explored a set of lemmatization python libraries (`Gensim`, `Spacy Lemmatizer` and `TextBlob`), being the `TextBlob` library the most efficient lemmatization technique found, since it considers the 'part-of-speech' (POS) tag for the word in a specific context and has a better treatment of plurals [5].

2.1.4.2 *Final features*.

After performing the exploratory data analysis, it has been identified new features that may be created, in addition to those variables described in section 1.2 . These are the following:

- % **Punctuation (expressed in times one):** Numeric variable that expresses the number of punctuations that each raw review has, divided by the number of words in each raw review.
- % **Stopwords (expressed in times one):** Numeric variable that expresses the number of stopwords that each raw review has before cleaning, divided by the number of words in each raw review.
- **Common Words Count:** Numeric variable that shows the number of top 20 words of each clean review.
- **Bigrams Count:** Numeric variable that shows the number of top 20 bigrams of each clean review.
- **Trigrams Count:** Numeric variable that shows the number of top 30 bigrams of each clean review.

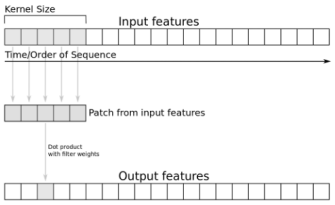
2.2. A CONVOLUTIONAL NEURAL NETWORK FOR NATURAL LANGUAGE PROCESSING.

[4] https://pypi.org/project/emoji/

[5] Plurals is a problem in some lemmatization libraries, for example, not all the final "s" should be considered as the plural form of a word. If this were done, will appear words that do not really exist, as is the case of `Pattern`, `Gensim` and `Standford` libraries.

2.2.1. *Convolutional Neural Network.*

Convolution starts by taking a patch of input features with the size of the filter kernel. With this patch you take the dot product of the multiplied weights of the filter.



1D Convolution

Now we are going to explain the different layers that we have included in our neural net. First of all, we have added an **embedding layer** to turn our sentences into word embeddings. To work with the embedding layer, after adding some convolutional and regularisation layers, we had to add a GlobalMaxPool1D layer which we will talk about it later.

Regularisation layers prevent over-fitting. **Dropout layer** is a regularisation layer which randomly set input units to 0 with a frequency of the rate introduced at each step during training time and it forces a neural network to learn more robust features. By adding this penalty, the model is trained such that it does not learn interdependent set of features weights. Gaussian penalties (l2) treat outliers a little more thoroughly, returning a larger error for those points. Those layers are added after any layer.

The **Convolutional layer** used is **Conv1D** and it is added twice with the same number of filters (256), the kernel size (5) and the activation function ("relu" for hidden layers). It creates a convolution kernel that is convolved with the layer input over a single spatial dimension to produce a tensor of outputs.

Then, the **GlobalMaxPool1D layer** is added. Downsamples the input by taking the maximum value of all features in the pool size. After this layer, a **dense layer** is added which connects neural network layer with all the weights and biases.

Last, it is necessary to add a dense layer with a **sigmoid** as the activation function because it is the output layer for our binary classification problem (see Figure 18).

After this brief outline of the characteristics of the model we have employed, now we will describe succinctly the training process and results.

2.2.2. *Variable selection and preprocessing.*

This model bases its predictive power exclusively in the patterns found in textual variables, that is why we are only considering this kind of variables, in particular *"ORIGINAL_PRODUCT_TITLE"*, *"ORIGINAL_REVIEW_TITLE"* and *"ORIGINAL_REVIEW"*, which we have merged into a new variable called *"fullReview"*. We take into account these three variables instead of only the one containing the review text in order to leverage the learning process of the model by adding more words to the corpus and syntactic relations to the embedding, which is needed as these kind of models require a vast amount of observations and large corpuses in order to boost their accuracy.

It also should be mentioned that the variables we are using to create *"fullReview"* are unaltered by any cleaning or lemmatizing method (that is why they contain "ORIGINAL" in their name), as we have realised that the model takes advantage of unlemmatized words, apostrophes in contracted words and numbers. Neither do we remove stopwords in this case due to the same reason, so we only apply to *"fullReview"* a basic preprocessing consisting in converting to lower case, removing symbols and remainders of HTML code, non breaking spaces and emojis. The **Tokenizer method** we use from Keras next also does a punctuation and symbol cleaning excepting apostrophes.

2.2.3. ***Vectorization.***

As mentioned before, the vectorization method employed to create the numeric vectors in which the model input consists is the Tokenizer from `keras.preprocessing.text`. The Tokenizer utility class can vectorize a text corpus into a list of integers. Each integer maps to a value in a dictionary (`tokenizer.word_index`) that encodes the entire corpus, with the keys in the dictionary being the vocabulary terms themselves. The parameter *num_ words sets* the size of the vocabulary where the most common *num_ words* words will be kept.

In order to achieve vectors with the same dimension despite the different length of the sentences, we need to perform a padding with zeros at the end of each sentence or to trim those sentences that are longer than the chosen dimension, which is 1500 so as to cast aside the minimum possible of information, as most of the sentences do not surpass that length.

2.2.4. ***Hyperparameter Tunning.***

The **hyperparameter tunning** is conducted by a Random Grid Search Cross Validation with four folds of twenty-five iterations, which means that twenty-five random combinations of the parameters set in the grid have been tested four times, obtaining the mean of the accuracy for each combination and returning the combination which shows the better performance in terms of mean accuracy.

2.2.5. ***Results.***

As shown in the graphics, the performance of the model trained with the resulting hyperparameters from the tunning is quite well balanced, having very similar performance to detect the true and the fake reviews. The measures taken to prevent over-fitting seem to have performed quite satisfactorily as well (see Figure 19).

2.3. MACHINE LEARNING MODELS.

The final dataset used is an ensemble between the original dataset and both output variables got with the convolutional neural net: one continue, *"DL_ CLASSIFICATION_ con"*; and the other one binary, *"DL_ CLASSIFICATION_ bin"*. There are several text data columns in our dataset such as *"Product_ Category"*, *"Review_ Tittle"*, or the most important feature *"REVIEW_ TEXT"*, which is the most extensive text data in our dataset.

In our research, we have tested several Machine Learning models to asses which one performs the best with the data we are handling, namely:

- Logistic Regression
- Random Forest
- Gradient Boosting
- XGBoost
- Support Vector Machine

Given these models, our task with each one of them has consisted of:

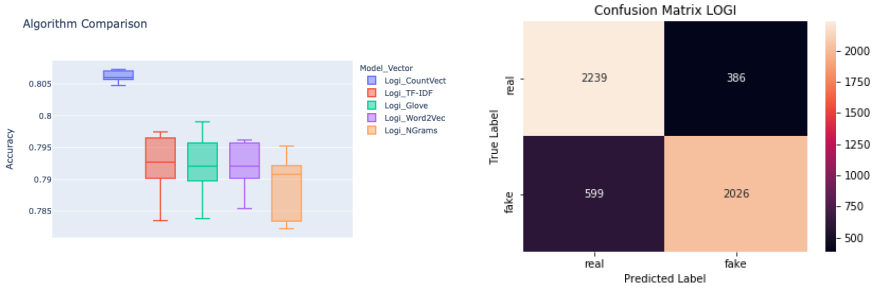
(1) Generating the feature vector which would serve as input of the model, comprising the vectorized form of the column *"REVIEW_TEXT"*, previously cleaned and lemmatized, and some other variables present in the dataset, some of them original, others originating from feature engineering. As we wanted to find out which one of the vectorization and embedding techniques we have already talked about in this paper offers the best results with each one of the mentioned models, we have proceeded to create five different feature vectors, one from each of these techniques to test, in particular: TF-IDF technique, Count Vectorizer technique, N-GRAMS technique, Word2Vec embedding and GloVe embedding, as we said before.

(2) Fine tunning each model by Grid Search with Cross Validation for each one of the five feature vectors we have generated.

(3) Comparing the results of the best tunned models for each vector among them, to find out which vectorization or embedding technique performs the best.

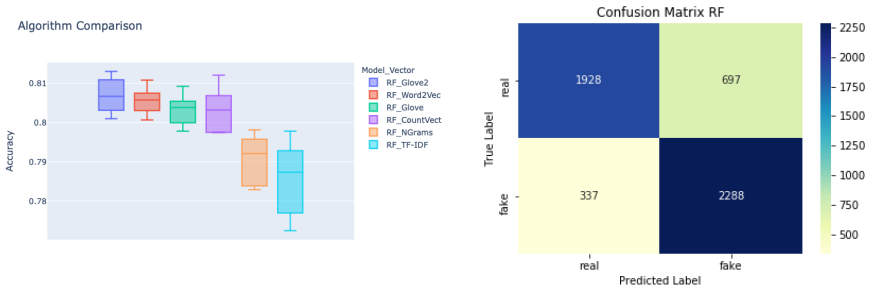
(4) Testing the resulting model from the best vectorization against the testing data to have a more realistic assessment of its performance. Their predictive capability can be showed using the confusion matrix where we condensed their test predictions along with real test labels. Here we have the summary of these results for each model:

(a) **Logistic Regression** (winner: Count Vectorizer vectorization technique):



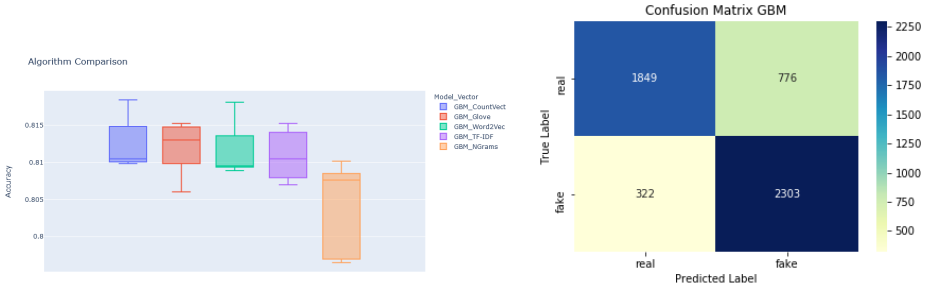
Logistic regression is a statistical algorithm used for binary classification problems. It is a predictive analysis algorithm and based on the concept of probability through the *logistic* or *sigmoid function*.

(b) **Random Forest** (winner: Glove embedding technique):



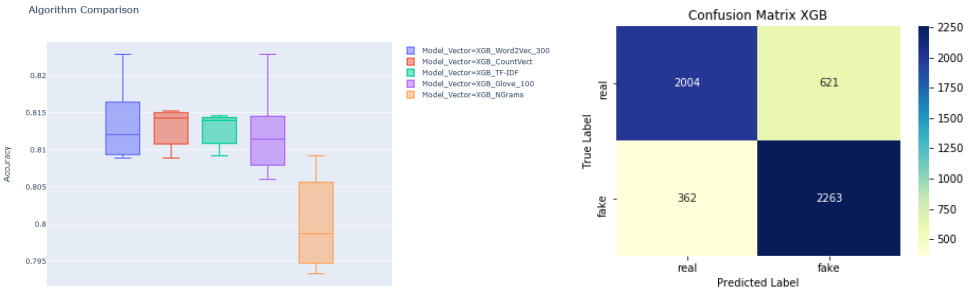
Random forest consists of a large number of individual decision trees that operate as an ensemble. Each individual tree splits out a class prediction and the class with the most votes becomes our model's prediction. A large number of uncorrelated trees allow getting a more accurate prediction. In order to get low correlation between trees, random forest lets each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is known as bagging.

(c) **Gradient Boosting** (winner: Count Vectorizer technique):



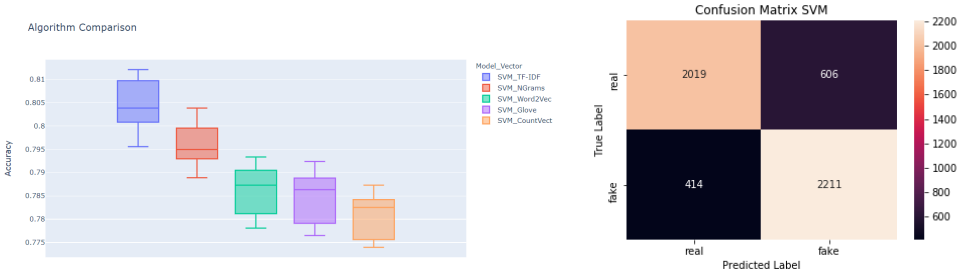
This method is named gradient boosting as it uses a gradient descent algorithm to minimise loss when adding models to the ensemble. It is based on the logistic function and the algorithm improves on each iteration by modifying the logistic function and getting a new better model. Stochastic gradient descent adds randomness by sampling observations and features in each stage, which is similar to the random forest algorithm except the sampling is done without replacement.

(d) **XGBoost** (winner: Word2Vec technique):



XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. It improves the GBM framework through systems optimisation and algorithmic enhancements using two regularisation parameters that penalise each tree's objective function of the GBM by the previous function.

(e) **Support Vector Machine** (winner: TF-IDF vectorization technique):



Support vector machines is a supervised learning algorithm which can be used for classification problems. It is based on the idea of finding a hyperplane (a function used to differentiate between features) that best separates the features into different domains.

2.4. **H2O AutoML.**

AutoML or Automatic Machine Learning is the process of automating algorithm selection, feature generation, hyperparameter tuning, iterative modeling, and model assessment. In our research, we have tried this H2O's noteworthy feature to compare its results against those obtained manually in the process described in this work, in order to discover new improvement areas in a continuous improvement work philosophy.

2.4.1. ***Some initial definitions.***

First, we needed to choose the input vector that AutoML is going to use in all the algorithms that will be tested. We have selected the one built by the Count Vectorizer technique since this is the technique that performs the best in three out of the five algorithms tested, two of them being the winners as we will explain in the conclusion.

H2O's AutoML automatically trains, tunes and cross-validates the following models within a user-specified time limit:

- a Random Forest (DRF)
- an Extremely-Randomized Forest (DRF/XRT)
- a random grid of Generalized Linear Models (GLM)
- a random grid of XGBoost (XGBoost)
- a random grid of Gradient Boosting Machines (GBM)
- a random grid of Deep Neural Nets (DeepLearning)
- 2 Stacked Ensembles, one of all the models and one of only the best models of each kind.

For now, we have excluded the stacked ensembles from this work, nevertheless, exploring them forms a part of our future research plans, as they can strongly leverage the predictive performance.

2.4.2. ***Results.***

We have defined that the winner model would be selected according to the higher performance in terms of AUC, (the results can be seen Table 7).

In the shown table we can observe that that the best algorithm is XGBoost, followed by GBM. These are the details of the winner model(*XGBoost_grid__1_AutoML_20200915_111018_model_3*):

- number_of_trees: 32.0
- ModelMetricsBinomial: xgboost
- MSE: 0.12911
- RMSE: 0.35933
- LogLoss: 0.41132
- Mean Per-Class Error: 0.17530
- AUC: 0.89391
- AUCPR: 0.88199
- Gini: 0.78783

2.4.3. ***H2O Segment Models - Training Segments.***

In H2O, you can perform bulk training on segments, or partitions, of the training set, which is a feature that we can take advantage of, since our data is segmented in categories of products, and each one of them can present specific traits where specialised models from the same algorithm (the winner) but with their own parametrization, can increase the predictive performance for each one of these categories.

To do this, the first step is to group similar categories among them. For now, we have conducted this segmentation in a manual way from the EDA inspection, resulting the following segments: Furniture, Baby, Others, Electronics, Music & Books, Health, Apparel, Jewelry, Sports, Home and Automotive.

As we advanced before, we have used our winner algorithms to perform this segmented training, that is to say: GLM (Generalized Linear Models) and XGBoost. There are up to 10 different models for each algorithm, which can be explored employing H2O's platform (H2O Flow).

For these trainings, we have left aside the text vectorizations, as we were interested in assessing how the other numerical features (the original ones and the ones from feature engineering) performed regarding category segmentation.

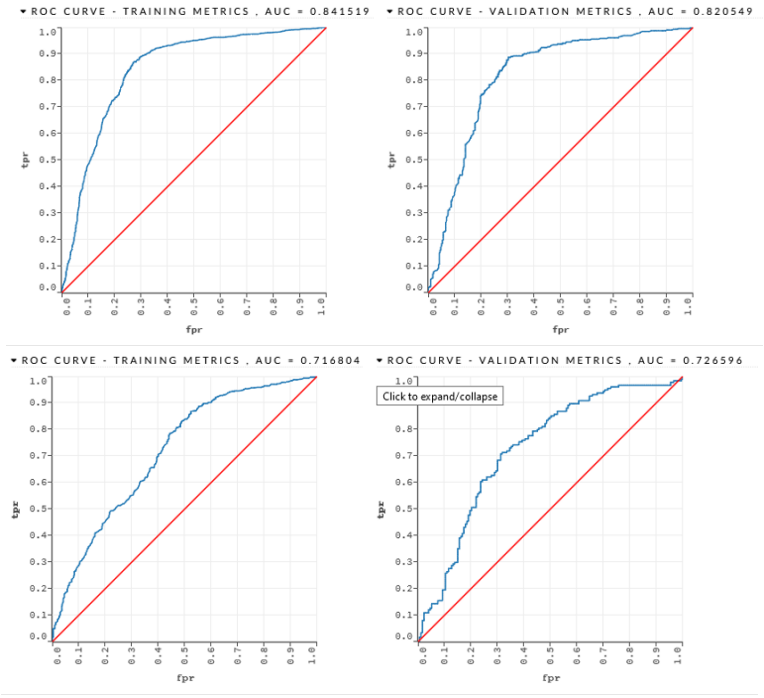
2.4.3.1 *GLM Results.*

As an illustration of the performance of this algorithm, we present the results for two of the above-mentioned categories: "Health" and "Others". Regarding Health category, the following graphic shows the coefficients for each of the input features, being *"VERIFIED_PURCHASE"* the most influential by far (see Figure 20).

If we observe the same graphic for Others category, we see that *"VERIFIED_PURCHASE"* keeps being the most relevant, but the rest of the features have changed places. This way we can notice that while "subjectivity" has a medium influence while reviewing health products, it is almost irrelevant in the "Others" category. The opposite happens to *"PUNCT"* feature, which is quite influential in Health category but has a low relevance in Others category. This supports the idea that a specific model for each category would be useful to enhance accuracy, since it gives the flexibility to adjust better without incurring in overfitting, as in production we would have a

vast amount of data for each category to train. The blue bars in the graphics refer to positive coefficients, while the orange ones refer to negative ones (see Figure 21).

The ROC curves for both train and validation set show an acceptably similar performance of the models in train and test, although it can vary significantly from one category to another, as the graphics for Health and Other categories respectively illustrate next:



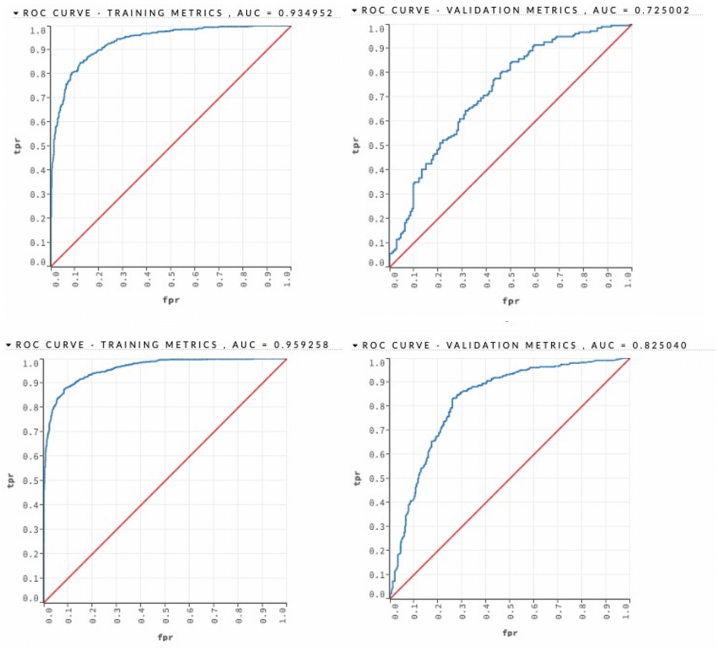
2.4.3.2 *XGBoost Results*.

Again, we will compare the same two categories, "Health" and "Others", to outline the performance of the XGBoost model.

Regarding Health category, the following graphic shows the importance of the variables, being *"VERIFIED_PURCHASE"* again the most relevant (see Figure 22).

If we observe the same graphic for Others category, we see that while *"VERIFIED_PURCHASE"* keeps being the most important, the rest of the features have also changed places. As these are variable importance graphics, there are no negative values, as it happened in the linear algorithms, which reflected the resulting coefficients of the model (see Figure 23).

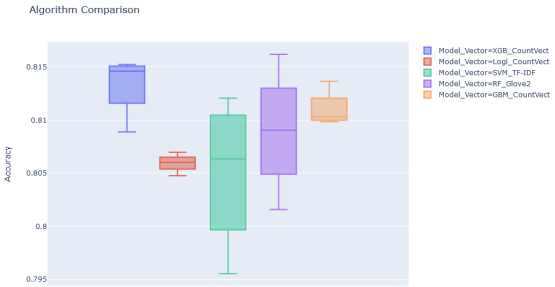
The ROC curves for both train and validation set show a quite different performance of the models in train and test, and also from one category to another, nonetheless, being the AUC for the testing data very similar to that achieved by GLM, as the graphics for Health and Other categories respectively expose next:



3. CONCLUSIONS

Observing the boxplot and the table comparing the best models of each algorithm among them, we reach the conclusion that we have to possible winners:

- XGBoost with Count Vectorizer: this is the model with the highest performance in terms of accuracy and AUC in the testing data. If the aim is to get the maximum possible accuracy, as in a competition, this would be the chosen model.
- Logistic Regression with Count Vectorizer: this model closely follows the former and has a quite low variance, with the added advantage that is much less computationally expensive and more friendly to extract conclusions about the variables from its resulting coefficients. Then, for a business environment, this could be the most appropriate model.



Model Vector	Accuracy Test	Precision Test	Recall Test	F1Score Test	ROC Curve	Params
XGB_CountVect	0.8127	0.8469	0.7634	0.8030	0.8762	'learning_rate': 0.1, 'max_depth': 15, 'min_child_weight': 10, 'n_estimators': 100
Logi_CountVect	0.8123	0.8399	0.7718	0.8044	0.8692	'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
SVM_TF-IDF	0.8057	0.8298	0.7691	0.7983	0.8686	'C': 1, 'kernel': 'linear'
RF_Glove2	0.8030	0.8512	0.7344	0.7885	0.8609	'max_depth': 15, 'max_features': 'auto', 'n_estimators': 200
GBM_CountVect	0.7908	0.8516	0.7043	0.7710	0.8514	'learning_rate': 0.1, 'max_depth': 11, 'n_estimators': 100

4. FUTURE RESEARCH

Having reached the end of this work, we have found some fields of study that would be interesting to delve into in future researches, such as the following:

- A deeper use of the tools that H2O provides regarding metrics and modelling.
- Exploring ensembled models, as they can significantly leverage the predictive power of our solution.
- Refining data segmentation by category employing ML models, since currently we are using a segmentation made by manual observation. Thus, the models by product category we talked about in the H2O section could be perfected.
- Exploring the possibilities that the newest pretrained models, such as GPT3 from OpenAI have to offer, for instance, to automate answers to comments in social media.

References

[1] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation, 2014.
[2] Nikolai Janakiev. Practical Text Classification With Python and Keras – Real Python.
[3] Amazon Reviews - kaggle.com
[4] Susan Li. Multi-Class Text Classification Model Comparison and Selection - Medium
[5] H2O - Open Source Leader in AI and ML

Annex: Figures

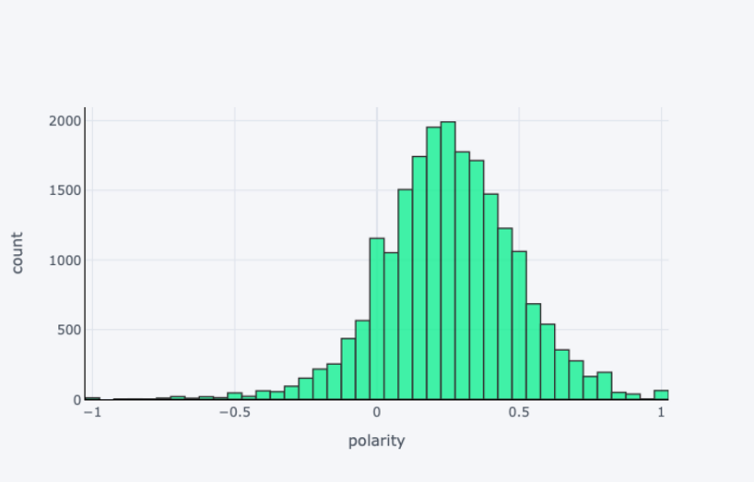


Figure 1. *Amazon Reviews Sentiment Polarity Distribution (continue)*
Note: Each bar represents the grade [-1,1] of polarity of each review.

Figure 2. *Review Rating Distribution (continue)*

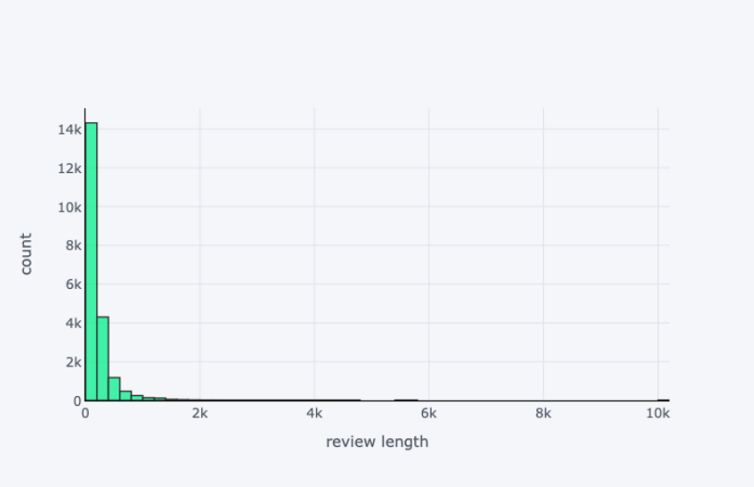


Figure 3. *Review Text Length Distribution (continue)*

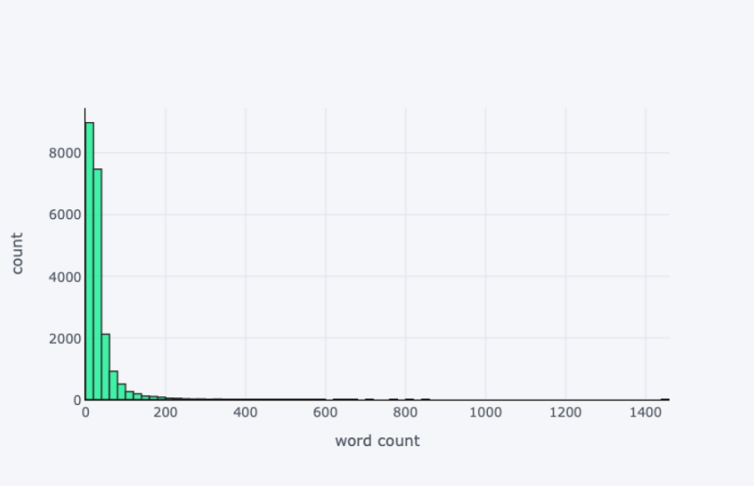


Figure 4. *Review Text Word Count Distribution (continue)*

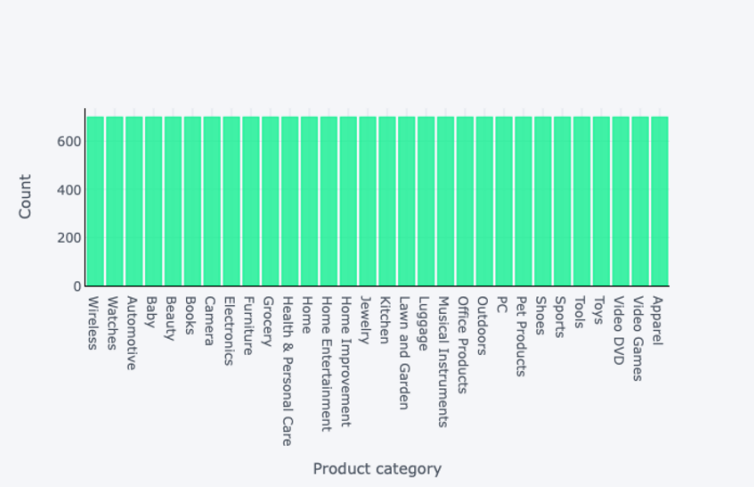


Figure 5. *Bar chart of Product Category (continue)*

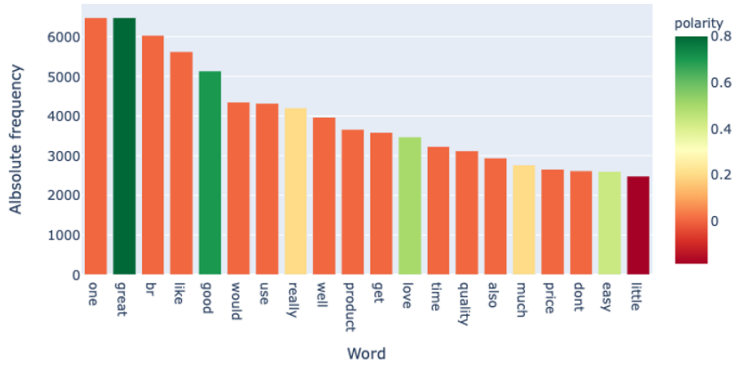


Figure 6. *Bar chart of top 20 words (continue)*

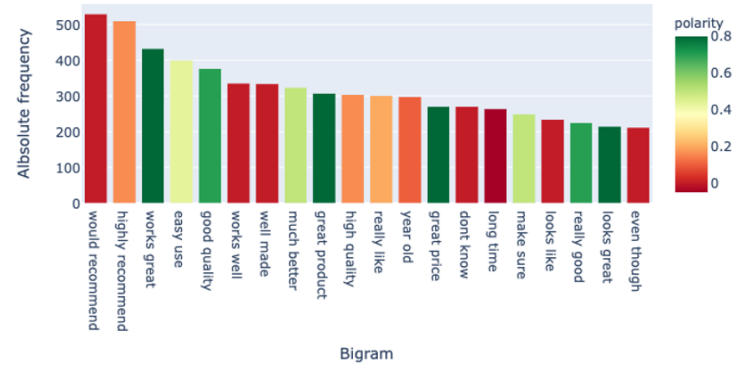


Figure 7. *Bar chart of top 20 bigrams (continue)*

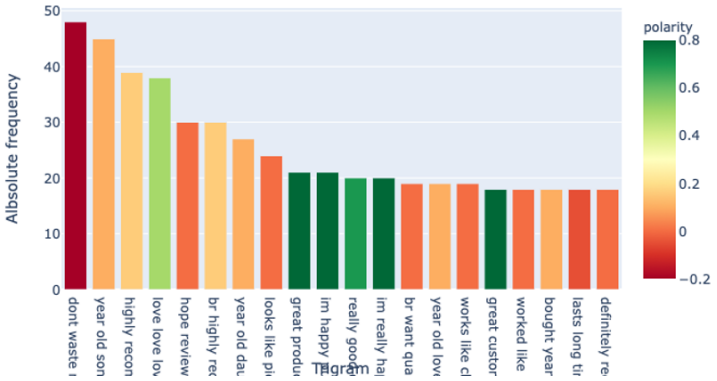


Figure 8. *Bar chart of top 20 trigrams (continue)*

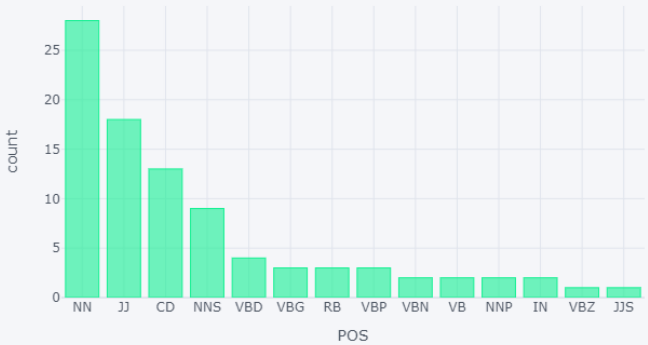


Figure 9. *Top 20 Part-of-speech tagging for the reviews (continue)*

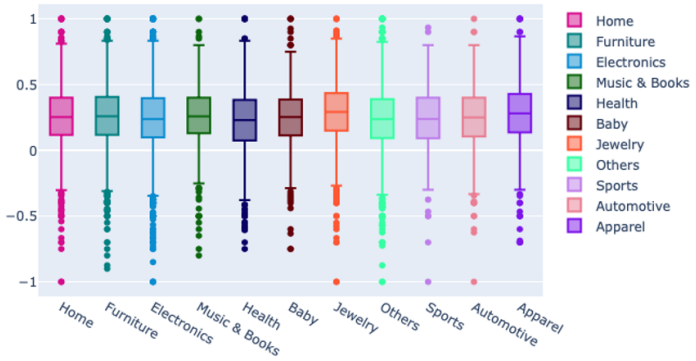


Figure 10. *Sentiment Polarity Boxplot of Product Category (continue)*

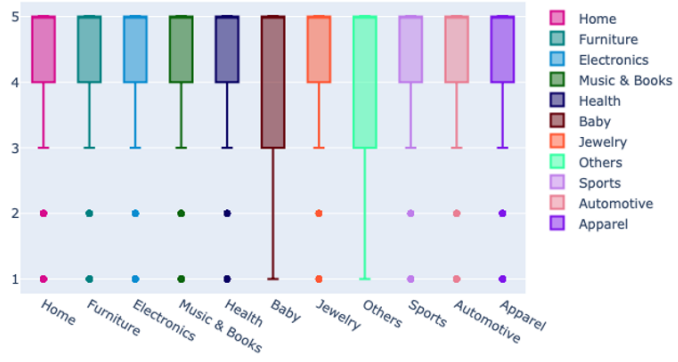


Figure 11. *Rating Boxplot of Product Category (continue)*

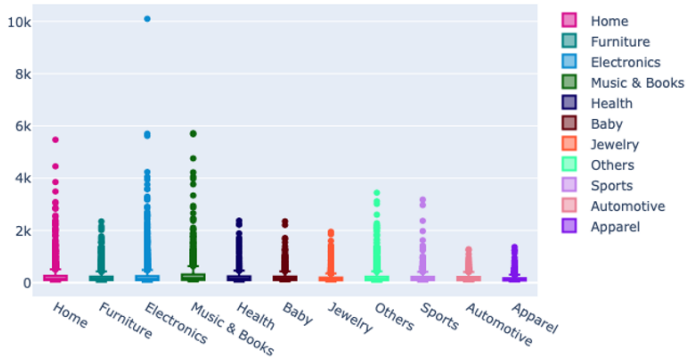


Figure 12. *Review length Boxplot of Product Category (continue)*

BIGRAMS:

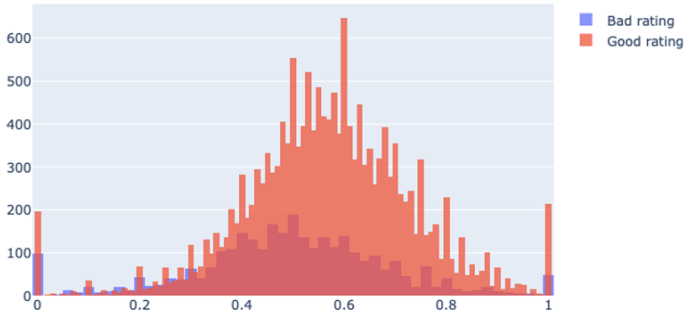


Figure 13. *Distribution of Sentiment polarity of reviews based on Good/Bad rating (continue)*

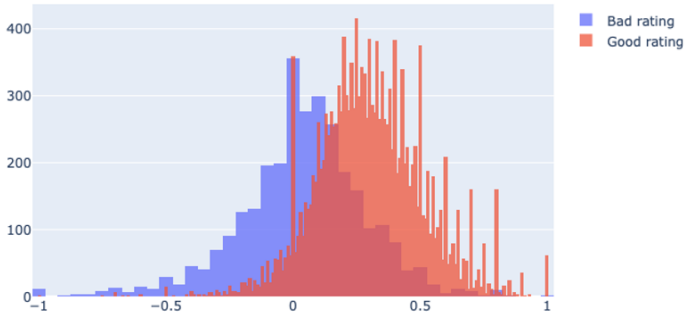


Figure 14. *Distribution of Sentiment Subjectivity of reviews based on Good/Bad rating (continue)*

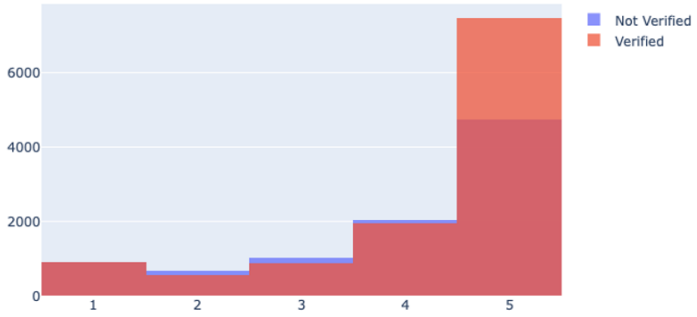


Figure 15. *Distribution of Rating of reviews based on Verified Purchase or not (continue)*

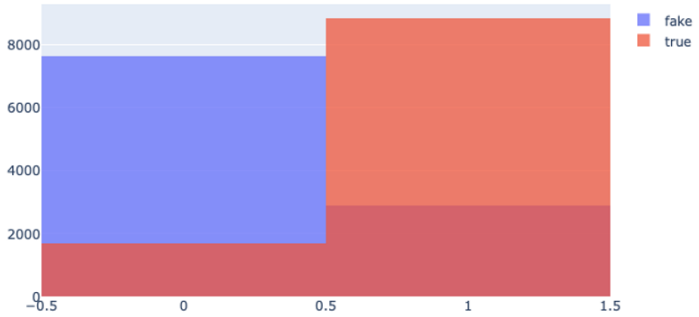


Figure 16. *Distribution of Fake or True reviews based on Verified Purchase or not (continue)*

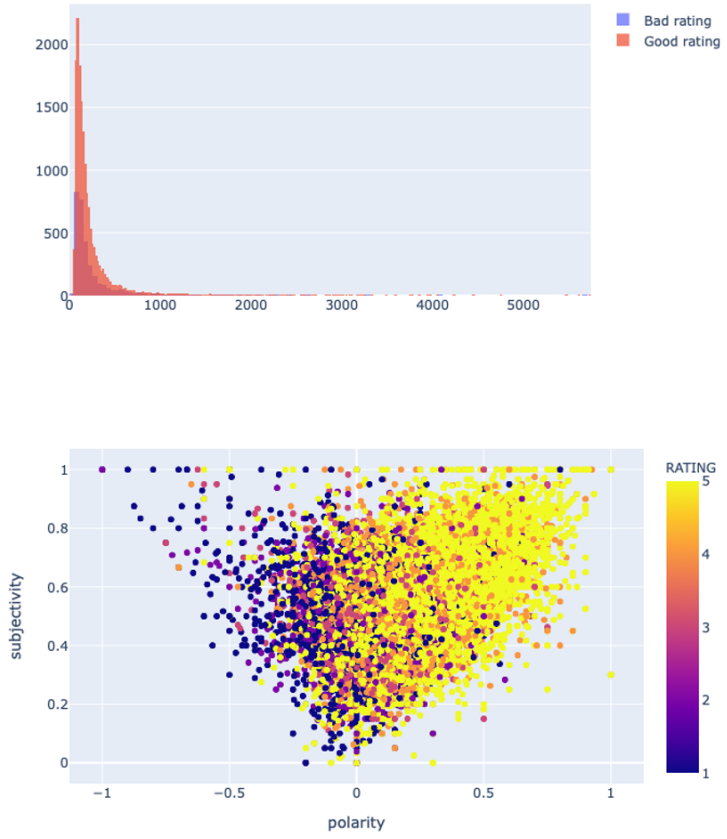


Figure 17. *Distribution of the rating of reviews based on review length (continue)*

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 1500, 300)         14082600
_________________________________________________________________
dropout (Dropout)            (None, 1500, 300)         0
_________________________________________________________________
conv1d (Conv1D)              (None, 375, 256)          384256
_________________________________________________________________
dropout_1 (Dropout)          (None, 375, 256)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 94, 256)           327936
_________________________________________________________________
dropout_2 (Dropout)          (None, 94, 256)           0
_________________________________________________________________
global_max_pooling1d (Global (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 512)               131584
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 14,926,889
Trainable params: 14,926,889
Non-trainable params: 0
_________________________________________________________________
```

Figure 18. *Model Summary (continue)*

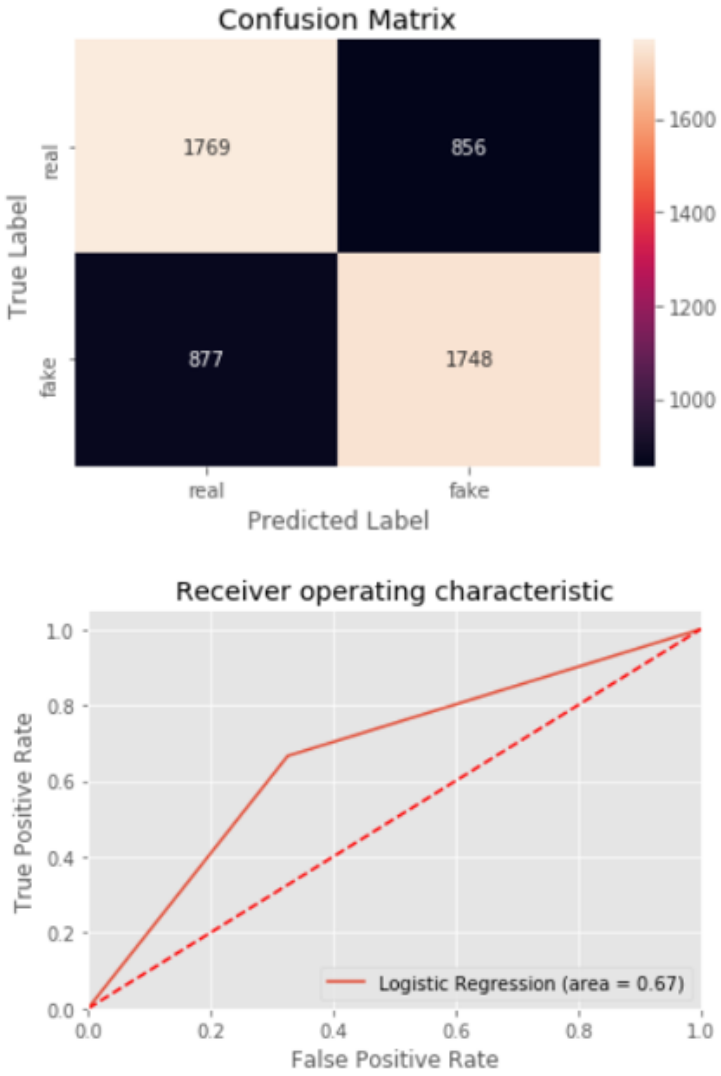


Figure 19. *Confusion Matrix and operating characteristics (continue)*

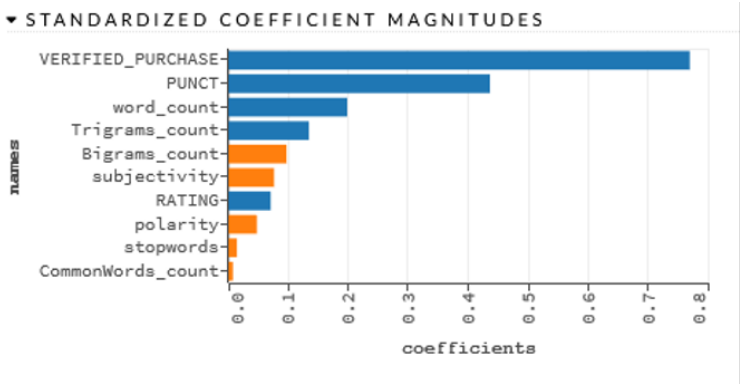


Figure 20. *Health Standardised coefficient magnitudes (continue)*

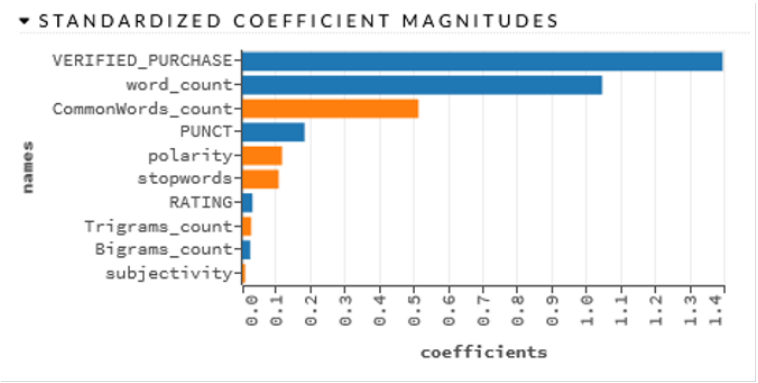


Figure 21. *Others Standardised coefficient magnitudes (continue)*

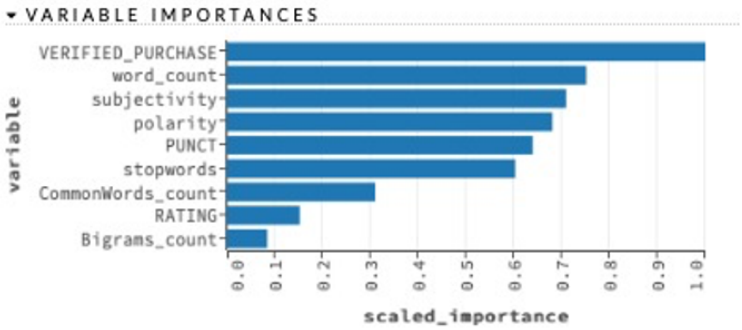


Figure 22. *Health Variable importances (continue)*

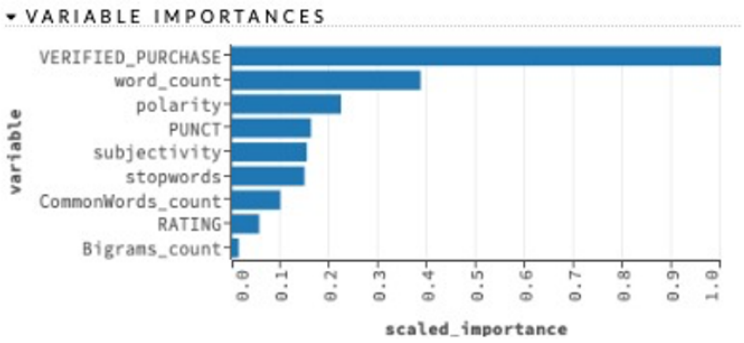


Figure 23. *Others Variable importances (continue)*

Annex: Tables

Word Index							
Document Index	000001	(...)	18000	28906	19071	(...)	35002
0	0	(...)	1	1	1	(...)	1
1	0	(...)	0	0	0	(...)	0
2	0	(...)	1	0	0	(...)	1
3	1	(...)	1	1	0	(...)	0
(...)	(...)	(...)	(...)	(...)	(...)	(...)	(...)
20997	0	(...)	1	1	1	(...)	1
20998	1	(...)	1	0	1	(...)	0
20999	0	(...)	0	1	1	(...)	0
21000	0	(...)	1	1	0	(...)	1

Table 1. Count Vectorization example (continue)

Word Index							
Document Index	000001 - 0003	(...)	18000 - 28765	28906 - 13456	03245 - 19071	(...)	00234 - 35002
0	0	(...)	1	1	1	(...)	1
1	0	(...)	0	0	0	(...)	0
2	0	(...)	1	0	0	(...)	1
3	1	(...)	1	1	0	(...)	0
(...)	(...)	(...)	(...)	(...)	(...)	(...)	(...)
20997	0	(...)	1	1	1	(...)	1
20998	1	(...)	1	0	1	(...)	0
20999	0	(...)	0	1	1	(...)	0
21000	0	(...)	1	1	0	(...)	1

Table 2. N-Grams example (continue)

Word Index							
Document Index	000001	(...)	18000	28906	19071	(...)	35002
0	0.12201	(...)	0.47695	0.20565	0.74920	(...)	0.00000
1	0.46909	(...)	0.35271	0.00000	0.51424	(...)	0.00000
2	0.67112	(...)	0.00000	0.48310	0.04115	(...)	0.00000
3	0.03748	(...)	0.34907	0.33357	0.83321	(...)	0.00000
(...)	0.00155	(...)	(...)	(...)	(...)	(...)	(...)
20997	0.07571	(...)	0.34131	0.54631	0.05471	(...)	0.26463
20998	0.22453	(...)	0.00000	0.00000	0.93501	(...)	0.24785
20999	0.00000	(...)	0.85536	0.93855	0.00000	(...)	0.12318
21000	0.67474	(...)	0.00000	0.96955	0.66464	(...)	0.00000

Table 3. TF-IDF example (continue)

Word	Frequency
one	6485
great	6484
br	6033
like	5623
good	5134
would	4346
use	4317
really	4204
well	3970
product	3660
get	3592
love	3469
time	3234
quality	3117
also	2943
much	2765
price	2668
dont	2618
easy	2604
little	2843

Table 4. Words Frequency (continue)

Bigrams	Frequency
would recommend	530
highly recommend	511
works great	433
easy use	400
good quality	377
works well	336
well made	334
much better	324
great product	307
high quality	304
really like	302
year old	298
great price	271
dont know	271
long time	264
make sure	250
looks like	235
really good	226
looks great	215
even though	212

Table 5. Bigrams Frequency (continue)

Trigram	Frequency
dont waste money	48
year old son	45
highly recommend product	39
love love love	38
hope review helpful	30
br highly recommend	30
year old daughter	27
looks like picture	24
great product great	21
in happy purchase	21
really good quality	20
in really happy	20
br want quality	19
year old loves	19
works like charm	19
great customer service	18
worked like charm	18
bought year old	18
lasts long time	18
definitely recommend product	18

Table 6. Trigrams Frequency (continue)

model_id	auc	logloss	aucpr	mean_per_class_error	rmse	mse
XGBoost_grid__1_AutoML_20200915_111018_model_3	0.8688	0.4456	0.8426	0.1949	0.3758	0.1412
XGBoost_grid__1_AutoML_20200915_111018_model_1	0.8666	0.4481	0.8365	0.1933	0.3772	0.1423
GBM_grid__1_AutoML_20200915_111018_model_1	0.8661	0.4538	0.8380	0.1953	0.3793	0.1439
XGBoost_3_AutoML_20200915_111018	0.8644	0.4507	0.8347	0.1958	0.3783	0.1431
XGBoost_2_AutoML_20200915_111018	0.8626	0.4566	0.8311	0.1985	0.3809	0.1451
XGBoost_1_AutoML_20200915_111018	0.8602	0.4595	0.8280	0.1991	0.3821	0.1460
GBM_grid__1_AutoML_20200915_111018_model_2	0.8597	0.4867	0.8272	0.2014	0.3934	0.1548
GBM_2_AutoML _20200915_111018	0.8588	0.4936	0.8201	0.2023	0.3965	0.1572
GBM_5_AutoML _20200915_111018	0.8584	0.5307	0.8199	0.2036	0.4151	0.1723
XGBoost_grid__1_AutoML_20200915_111018_model_4	0.8581	0.4725	0.8274	0.2022	0.3863	0.1493
GBM_1_AutoML _20200915_111018	0.8575	0.5090	0.8184	0.2034	0.4043	0.1634
GBM_3_AutoML _20200915_111018	0.8573	0.5199	0.8198	0.1975	0.4097	0.1679
GBM_4_AutoML _20200915_111018	0.8529	0.5501	0.8125	0.2018	0.4256	0.1811
GLM_1_AutoML _20200915_111018	0.8526	0.4737	0.8234	0.2000	0.3868	0.1496
XGBoost_grid__1_AutoML_20200915_111018_model_DeepLearning_1_AutoML_20200915_111018	0.8516 0.8420	0.4934 0.5095	0.8206 0.8078	0.2121 0.2081	0.3935 0.3969	0.1548 0.1576
DRF_1_AutoML _20200915_111018	0.8364	0.5092	0.7952	0.2179	0.4037	0.1630
DeepLearning_grid__1_AutoML_20200915_111018_model_1	0.8268	0.8684	0.7861	0.2200	0.4251	0.1807
DeepLearning_grid__2_AutoML_20200915_111018_model_1	0.8206	0.79100	0.7838	0.2292	0.4352	0.1894
XRT_1_AutoML _20200915_111018	0.8187	0.5361	0.7749	0.2288	0.4203	0.1766

Table 7. Models summary (continue)