A Text Mining Framework for Big Data

Niki Pavlopoulou, Aeham Abushwashi and Vittorio Scibetta

Abstract Text Mining is the ability to gain insight from text. This is a challenging task, especially when the target is large amounts of data. Big Data has attracted much attention lately both from academia and industry. A number of distributed databases, search engines and frameworks have been developed to handle the memory and time constraints, which processing large amount of data requires. However, there is no open-source end-to-end framework that can combine real-time and batch processing of ingested big textual data along with some user-defined options and provide specific, reliable insight from the data. This is important as in this way new unstructured information is made accessible in real-time, more personalised customer products can be created, novel unusual patterns can be found and actioned quickly. This work focuses on a complete real-time automated classification framework of unstructured data with the use of Natural Language Processing and Machine Learning algorithms on Apache Spark. According to our findings, our Spark-based platform performs as well as the best experiments done on the publicly-available 20NewsGroups dataset, both in terms of throughput and memory utilisation.

1 Introduction

The problem of automatically classifying unknown text documents is an important one. The amount of text nowadays is vast originating out of sources ranging from

Niki Pavlopoulou
Department of Computer Science, University of Reading, Whiteknights, Reading, Berkshire, RG6 6UA, UK, e-mail: n.pavlopoulou@reading.ac.uk

Aeham Abushwashi
Exonar, 14 W Mills, Newbury, Berkshire, RG14 5HG, UK e-mail: aeham.abushwashi@exonar.com

Vittorio Scibetta
Exonar, 14 W Mills, Newbury, Berkshire, RG14 5HG, UK e-mail: vittorio.scibetta@exonar.com

the World Wide Web, social media, e-mails, medical records, databases etc. Much of this data inherently lacks coherent structure within. There is far too much data for human users to manually go through and categorise. Therefore, methods like Text Mining have emerged to solve this issue [47].

Classification in Text Mining is the ability to attach a label to a previously unseen text object according to models extracted from a collection of labeled texts [37]. The acquisition of such a collection that is called training set is a challenging matter. Most of the time this set is manually created by subject matter experts who identify the best documents that represent each label. This process can often be biased and time-consuming and there is also no specific target for the amount of documents this set should have. The more the better is not true for all cases [21].

After the training set is gathered, a number of pre-processing techniques are used [3]. Often one cannot know which method or combination of methods will work best and blindly tries to use some of them to see how good their results are. Evaluation techniques, such as k-fold cross validation and holdout, are typically used to help optimise the selection process.

Once the input data has been preprocessed, Machine Learning (ML) algorithms are used in order to build models that can learn from the training set and that can later be used to classify previously unseen documents [3]. There is a plethora of algorithms to do this and one can determine through experimentation which one or ones work best for a specific problem.

There are a number of frameworks and tools for Data Mining [7, 40], but most are not open-source. RapidMiner [26], Orange [13], KNIME [5], Weka [17] are on the other hand some of the most popular open-source tools that can support a plethora of algorithms, pre-processing methods and rich GUIs. However, they do not cope with large amounts of data nor do most of them have support for stream processing. Whilst these tools and framework are perfectly usable in environments where data volume is low and streaming capability is not required, these constraints tend to limit their applications in industry.

With rising interest in real-time data processing, a lot of attention is being paid to streaming big data frameworks like Apache Spark [25], Apache Storm [2] and Apache Flink [9]. Others, like Azure ML [4], SAMOA [12] and TensorFlow [1] are also very powerful tools. However, Azure ML supports cloud services only, which blocks out users who need the capability on premise. SAMOA, on the other hand, supports only streaming data, which is not applicable for all applications. TensorFlow supports Deep Learning, where classic ML techniques can be easily used to solve the same problems that industry has.

We focused our attention on Apache Spark, which is widely used in production by many organisations and it is actively maintained and extended to support an ever-increasing list of use cases. Although Spark is a powerful framework with good support for a number of ML algorithms, it lacks support for pre-processing methods, native capability to ingest data from different types of data sources and the ability to store data in databases. In order for the platform to be viable for a large-class of applications, a more comprehensive workflow is required. This workflow should

ingest data from various sources, pre-process it, use existing or newly-implemented algorithms, evaluate and visualise results.

This paper addresses the task of building a system that can take advantage of the batch and streaming capabilities of Apache Spark for a complete Text Mining application. In order to do this, our method uses a variety of tools, libraries and the capabilities underlying the Exonar platform.

The paper is organised as follows; Section 2 describes the methods and the architecture of the analytics platform, Section 3 provides results on the 20NewsGroups dataset and Section 4 describes conclusions and future work.

2 Methods

Our platform consists of two processes: the model building and the ongoing prediction. In order for these processes to take place a number of stages need to be addressed.

2.1 Dataset Collection

The first step in model building is the creation of the training set. In our case, this is created using the Exonar search and discovery platform and it can span from publicly available data to client data. The product collects and analyses enterprise data and stores it in HBase [14], a NoSQL database, and enables users to easily identify training sets for a given classification task.

2.2 Text preprocessing

Raw text is not suitable as direct input to a classification engine. The text needs to be represented in a way that is useful enough to be fed as an input into a ML algorithm. There are a number of techniques that can be used [37] for this purpose. Below are some of the most prominent ones that are implemented on our platform and an advanced user has the ability to choose from. The implementation is done with the combination of in-house methods, Spark, Apache Lucene [6] and OpenNLP [20].

2.2.1 Document representation

A collection of documents D contains a number of unique words. These words represent the dictionary. The dictionary is mapped to integers that now represent each unique word. This reduces pressure on memory, which is utilised heavily anyway,

during the execution of the resource-intensive ML algorithms. A document d_i is then represented as a vector, where each index corresponds to its unique mapped word and each value is a weight of this word in the document according to a weighting scheme. Feature extraction techniques are used to analyse the original words and end up with a concise and more representative dictionary in the end.

2.2.2 Feature extraction

Texts in general may contain lots of ambiguity, syntactical errors, high occurrences of certain words or semantic similarity. It helps to separate words that are syntactically similar but semantically different, as does grouping words that are semantically the same. For example, the word ”bank” in the concept ”Bank of England” or ”bank of Thames” is different. Conversely, ”monkeys” and ”apes” are under the umbrella of ”primates”. In order to pinpoint these events, some filters are used and their results are shown in Fig. 1.

Tokenisation: transforms a document d_i to a collection of words (w_1, w_2, .. , w_n), often called ”bag of words”. Simple tokenisers, like whitespace tokeniser, are used in addition to some specialised tokenisers that can capture specific types of entities, such as URLs.

Lowercase filter: transforms all words into lowercase characters. In this way, words like ”Runner” or ”runner” can be identified as the one token, that of ”runner”.

Stopword filter: removes words like ”a”, ”the” etc. that may be deemed semantically insignificant.

Stemming: transforms words into their root form. For example, words like ”connection”, ”connecting”, ”connector” are transformed to the root ”connect”.

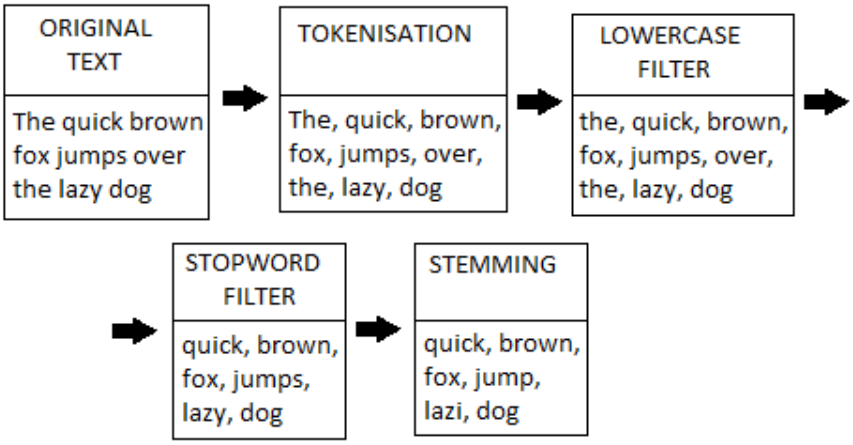


Fig. 1: Stages of feature extraction from text

2.2.3 Feature selection

One of the main challenges in Text Mining is high-dimensionality. Texts often contain lots of words that can range from very important to highly noisy. Different data sets can, and do, have wildly different characteristics. It is useful in building a general purpose text classification system to allow users to choose which features are valuable enough to be kept in the dictionary. The techniques that are used range from option tuning to statistics.

One option could be to keep all features. That would be wise if necessary, because this will be memory-heavy and the ML algorithms will take much more time to process the data. Another option would be to discard analysed words of length $\leq$ 2 or non-analysed words of length ≤ 3, because chances are they might not prove significant enough.

Another option could be to keep the k most important words, where k is user-defined according to evaluation tests. The importance of the word is defined according to its weight, that is the higher the weight the more important the word.

Words that are extremely rare throughout the collection D may be noise. Having an option of eliminating those words that exist in less than x documents, where x is user-defined, will result in removing those rare words and possibly providing more accurate results.

Information gain [23] is a statistical method used by Decision Tree (DT) and Random Forest (RF) for selecting the most significant features throughout the dictionary. A sample si of the collection D is related to those documents that belong to a specific class ci. A word wi may be contained in this sample or not. Entropy H depicts how homogeneous the si is. The lower the entropy the higher the homogeneity. Therefore, the information gain of a word wi for a sample si is:

$$\begin{aligned} IG(w_i;s_i) = H(s_i) - H(s_i|w_i) = -\sum_k p(C_k)\log p(C_k) \\ + p(wi)\sum_k p(C_k|w_i)\log p(C_k|w_i) + p(w_i')\sum_k p(C_k|w_i')\log p(C_k|w_i'), \end{aligned} \tag{1}$$

where p(wi) is the probability of wi occurring, p(w'i) is the probability of wi not occurring, p(Ck) is the probability of the k-th class, p(Ck|wi) is the probability of the k-th class given the occurrence of wi and p(Ck|w'i) is the probability of the k-th class given the non-occurrence of wi. As the algorithm traverses all possible words, the ones that have the highest information gain will be selected in the end.

Chi-square [44] is a statistical method that examines the dependency between a word wi and a class ck. It is defined as:

$$\chi^2(t,c) = \frac{N(AD-CB)^2}{(A+C)(B+D)(A+B)(C+D)} \tag{2}$$

$$\chi^2_{avg}(t) = \sum_{i=1}^{m} Pr(c_i)\chi^2(t,c_i) \tag{3}$$

$$\chi^2_{max}(t) = \max_{i=1}^{m}[\chi^2(t, c_i)], \tag{4}$$

, where |D| is the number of documents in the collection D, A is the number of times wi occurs in documents belonging to class ck, B is the number of times wi occurs in documents that do not belong to class ck, C is the number of times ck occurs without the word wi and D is the number of times neither ck nor wi occurs. The higher the chi-square the more dependent wi and ck are. Therefore, a selection of the k most dependent words for each class are selected in the end, where k is user-defined.

2.2.4 Feature representation

The most highly used feature representation model is the bag-of-words or unigrams. This model is the case, where the dictionary consists of words of length one. However, there are times when the meaning of words, the sequence of them and the phrases that exist play a vital role [41]. Providing only individual words to the ML algorithm in this scenario will produce less accurate results. For example, if the text contains the words ”white house”, then a unigram would split the two words into ”white” and ”house”, whereas a bigram would retain the token ”white house”. It is important to evaluate the different feature representations and identify the one that yields better results. The user can either use uniGrams or nGrams of $n > 1$, where n is user-defined.

When using nGrams it is sometimes important not to proceed with stopword removal or stemming. For example, if stopwords were removed from the word ”state-of-the-art” then the words ”of” and ”the” would be eliminated resulting in a bigram of ”state art”, which has a different meaning. Stemming could also alter the original meaning of phrases. However, nGrams are memory-intensive as more words are created than with uniGrams, so sometimes the stopword removal and stemming could prove useful for the time performance of the ML algorithm without even altering the final accuracy. The choice of using stopword removal or stemming for nGrams is user-defined, because its usefulness is ultimately dependent on the dataset.

Part-of-speech (POS) tagging is another method that is widely used for representing features. Texts are as dynamic as the language they are written in. The same words could have a totally different concept when used in a specific syntactical way. For example, ”my dog is barking” and ”the bark of the tree” refer to the word ”bark”, which is a verb in one context and a noun in the other. With POS tagging, the first text segment results in ”bark[VERB]” and the second text segment results in ”bark[NOUN]”. This means that the uniqueness of the word between these texts is retained. Therefore, the user has the ability to choose either to use POS tagging or not, according to evaluation tests.

2.2.5 Weighting scheme

As explained above, each document is represented by a vector with indices, which represent the mapped words and values of their corresponding weight. This weight derives from a number of weighting schemes with the most popular one being the term frequency(TF) inverse document frequency(IDF) metric.

TF-IDF captures the uniqueness and significance of each word wi in a collection D and enables the related documents to be identified. TF is often the number of times a word is seen in a document, nevertheless, we have tried other TF metrics, like log normalisation and double normalisation to avoid introducing bias due to the length of the document. Log normalisation was found to be the generally most acceptable one for our cases. The TF-IDF score is calculated as:

$$TF = 1 + \log(f_{t,d}) \tag{5}$$

$$IDF(t,D) = \log \frac{|D|+1}{DF(t,D)+1} \tag{6}$$

$$TFIDF = TF \times IDF, \tag{7}$$

where ft,d is the frequency of the word t in the document d, |D| is the number of documents, DF(t, D) is the number of documents that contain the word t.

The concept behind this metric is that the more a word is used in a document, the more representative it is for this document. However, the more the term is used in collection D, the less discriminative it is.

2.2.6 Avoiding bias

Often a dataset will contain a number of bias-inducing documents. Bias is a challenging factor that can result in overfitting a model, that is the model may learn very well from a dataset, but it might fail to generalise for new data. Algorithms such as DT and RF that rely on information gain for feature selection can overfit significantly when trained on biased data. Therefore, specific features need to be eliminated in order to avoid this bias when the dataset is created. It is important the dataset is balanced enough for all classes. If the dataset contains too many documents of one class and quite a few for another, then a model is bound to be biased towards the class with the most documents, hence inaccurate results will occur. In our platform if there is sufficient data for all classes then balancing is done automatically.

2.3 Evaluation

Once the training set has been pre-processed, it is fed into either an evaluation process or a model building process with default algorithmic parameters. The algorithm, its parameters and whether evaluation is required or not are defined by the user through the configuration file. In general, during evaluation the training set D is split into a subset of the training set D' and a test set T. D' is used to build the model and T to predict the documents. There are two types of evaluation that are supported by our platform, k-fold cross validation and holdout validation, with the choice of which one is used being configurable. Once the user selects the evaluation to take place, the evaluation metrics of all classes and averaged ones are stored.The implementation has been done from scratch on Spark.

The evaluation methods and the metrics used are defined below.

2.3.1 K-fold cross validation

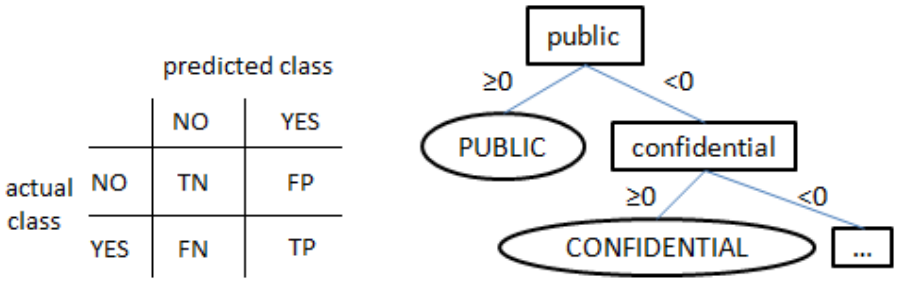
This method splits D randomly into k mutually exclusive subsets of equal size, which are called folds. Then an iteration procedure begins, where in the 1st round the 1st fold is used as T and the rest as D, in the 2nd round the 2nd fold is used as T and the rest as D and so on until each fold has been used as T exactly once. The final evaluation metrics consists of the average value of the metrics of all rounds. This is considered to be the best evaluation method, although it can take a considerable amount of time depending on the data volume and the algorithm that is used. K=10 is often considered to be the best value [35].

2.3.2 Holdout

This method splits D into 70% as D' and 30% as T. The method is not considered to be as accurate as the k-fold cross validation, however it is useful when there is a large amount of data and the user needs fast and approximate evaluation metrics [19].

2.3.3 Evaluation metrics

Since T contains the real classes, one is able to estimate how well the model behaves by comparing the true classes versus the predicted ones. A confusion matrix [32] shown in Fig. 2a is useful in this case, which contains all true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). In a binary classification problem of whether a class is positive or negative, TP is the members of the class that were predicted as such, TN is the non-members of the class that were predicted as such, FP is the non-members of the class that were predicted as members



(a) Confusion matrix

(b) A DT of a binary classification problem PUBLIC vs CONFIDENTIAL

Fig. 2: Confusion Matrix and a DT

of the class and FN is the members of the class that were predicted as non-members of the class.

There are specific evaluation metrics that can be used to cater for the model's performance and are described below.

$$Accuracy = \frac{TP+TN}{total} \tag{8}$$

Accuracy is the proportion of documents that were predicted correctly.

$$Precision = PPV = \frac{TP}{FP+TP} \tag{9}$$

Precision is the proportion of the predicted positives that are indeed positives.

$$Recall = TPR = sensitivity = \frac{TP}{FN+TP} \tag{10}$$

Recall is the proportion of positives that are correctly predicted as such.

$$F-measure = 2 \times \frac{precision * recall}{precision + recall} \tag{11}$$

F-measure is the weighted harmonic mean of precision and recall

$$FPR = fall-out = \frac{FP}{TN+FP} = 1 - specificity \tag{12}$$

FPR is the proportion of negatives that are predicted as positives

In an ideal case scenario, the user would expect the first four metrics to hit 100% and the last 0%.

The user has the ability of not using evaluation and proceed into building a model with default settings.

2.4 ML Algorithms

Once the evaluation process is done, the platform picks up the best combination of parameters and builds a model, otherwise a model is built according to default parameters which work well for most cases. The model is then saved in a distributed data store.

A number of algorithms have been evaluated during the course of the project and it has been decided to support a few of these, namely DT, RF, Support Vector Machines (SVM) and Naive Bayes (NB) that are available on Spark.

2.4.1 DT

A DT [37] shown in Fig. 2b is a tree-structured model that can predict the class of a new document. The algorithm uses information gain to select the most important features of a collection D and creates a binary tree, where each leaf is a class and each branch is a feature with a TF-IDF weight threshold. The root of the tree is the most significant feature followed by the less significant features until a user-defined depth is reached. During model building and for each branch level, the algorithm checks if all documents that apply to this branch belong to the same class. If they do then a leaf is created with this class, if they do not then more features are examined that satisfy the branch to create new branches etc. The prediction of a new document can be done when its features traverse the tree from the root until a leaf is reached, which represents the class of the document. A DT is easily interpretable by users, but its main disadvantage is that it can easily overfit.

2.4.2 RF

A RF [8, 24, 38] is a collection of DTs. The main difference with DTs is that each tree is created by a method called bootstrapping, which is a random selection of the original training set D and the best features are selected by different random subsets of the features. The prediction of a new document is the majority prediction class across all trees that the document's features have traversed. The advantage of RF is that it is less prone to overfitting than DT, but it is much more costly especially if the number of trees is high. The user can select the number of trees and the depth of each tree.

2.4.3 SVM

SVM [18, 27] is a linear model for binary classification that tries to find the maximum margin hyperplane (MMH) that best separates the two classes. Documents that are closest to the MMH are called support vectors.

The SVM satisfies the following formula:

$$\begin{aligned} w^T \phi(x_i) + b \geq +1, \quad if \quad y_i = +1 \\ w^T \phi(x_i) + b \leq -1, \quad if \quad y_i = -1 \end{aligned} \tag{13}$$

which is equivalent to:

$$y_i[w^T \phi(x_i) + b] \geq 1, \quad i = 1, ..., N, \tag{14}$$

where xi the training set and yi the target class, w: weight and b: bias, ϕ = x in our linear case.

Eq.14 then is responsible for the construction of two hyperplanes that are separated by the hyperplane:

$$w^T \phi(x) + b = 0 \tag{15}$$

in the feature space. The margin width of the two hyperplanes is:

$$\frac{2}{\|w\|^2} \tag{16}$$

During prediction a document will be subjected to the formula:

$$sgn(w^T \phi(x) + b) \tag{17}$$

The optimisation problem is defined as:

$$\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i \tag{18}$$

subject to

$$\begin{aligned} y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \quad i = 1, ..., N \\ \xi_i \geq 0, \quad i = 1, ..., N \end{aligned} \tag{19}$$

In order to cater for multi-class models, our platform creates a model for each class by considering the corresponding class as the positive one and the rest as the negative ones (one-versus-all classifier). Then once a new document collects its prediction for all models, the one that responds to the positive one will win.

One of the main advantages is that SVM can reach a global optimum and overfitting has less chances to occur. It has been shown that is one of the most successful classification algorithms used [3] and it can handle high-dimensional data. However, one of the main issues is the computational cost, especially for non-linear SVM and the parameter selection when non-linear SVM takes place [10, 11]. The user can select the number of iterations the algorithm will use to reach the minimum in Eq.18.

2.4.4 NB

NB is a probabilistic model that is based on Bayes' Theorem. It is called naive, because it observes all features as independent to one another, which is not the case for texts in general. For example, it assumes that the order of the features or the existence of features to the same text play no significant role. On the other hand, it is one of the fastest ML algorithms in terms of classification.

In our platform the multinomial NB [34] version is used. The likelihood of a document is defined as:

$$p(d \mid \theta'_c) = \frac{(\sum_i f_i)!}{\prod_i f_i!} \prod_i (\theta_{ci})^{f_i}, \tag{20}$$

where c: class, θ'c: parameter vector of class c with values of θci that is probability that feature i occurs in class c, fi: the frequency count of feature i in document d.

The predicted class will be the one with the highest posterior probability, which is defined as:

$$\begin{aligned} l(d) &= argmax_c[\log p(\theta'_c) + \sum_i f_i \log \theta_{ci}] \\ &= argmax_c[b_c + \sum_i f_i w_{ci}], \end{aligned} \tag{21}$$

where bc: threshold term, wci: class c weight for feature i.

The user can select the lambda parameter, which is responsible for additive smoothing in case one encounters words that did not exist in a training set or that do not appear in a specific class, in order to avoid conditional probabilities of resulting in 0.

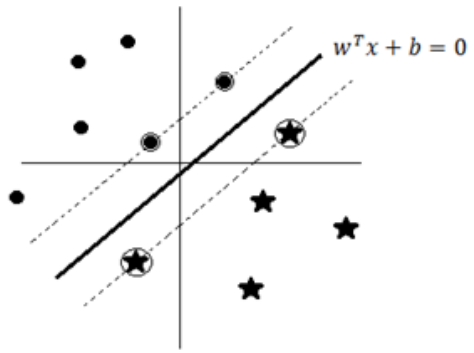


Fig. 3 An SVM for a binary classification of stars versus circles that are linearly separable. The thick black line is the optimal hyperplane and the circled points in the dashed line are the support vectors.

2.5 Multi-type model building

The Exonar platform supports processing data in near real-time. Consequently, the classification engine must support two modes of operation for model building; batch-processing of existing data and on-demand for new data as it is ingested. The capabilities of Spark are suited for both modes. Creating new models from fresh data is essential, because new data might have highly related topics that are themselves unrelated to old data; therefore, existing models will be of no use. Furthermore, streaming could be also used for real-time user demands of different model buildings. For example, a number of users might need to build different models for different training sets or cases; therefore these requests could be streamed in real-time in order to cater for these demands.

2.6 Real-time prediction

After the model building phase is complete, the prediction process can start. It is essential for the prediction to be very fast and allow users to classify previously unseen documents in near real-time. The user defines through a configuration file, which model needs to be used. User requests are queued in a message queueing service, the data is collected and the same pre-processing procedure takes place as the one applied during the model building process. Then the user-defined model is read from the distributed data store and is used to classify these documents. In the end, the user sees only the label of the document and a confidence score,which gives an indication of how valid the prediction is. This whole process was implemented from scratch in Spark.

The confidence scores are essential to create a measuring stick and use it to establish confidence in the quality of predictions. We observed that some ML algorithms can be well-calibrated, that is the prediction probability is indeed the confidence score, but there are others that perform poorly in this regard. Other algorithms do not support extraction of confidence score at all. There are many studies [28, 31, 36, 39, 43, 45, 46] that addressed this issue.

2.6.1 DT confidence score

DTs create tree-structured models, where each leaf contains a class. The creation of the leaf means that the majority or all of the instances that fall under this leaf belong to the leaf's class. Therefore:

$$P = \frac{k}{n}, \tag{22}$$

where k: positive instances of the leaf's class and n: all instances under the leaf.

Studies [45] suggest that this score is biased, since DT tend to create homogeneous leaves or statistically unreliable when the instances that fall under a leaf are small in numbers.

2.6.2 RF confidence score

RF is a collection of DT, where each of the tree creates a confidence score, as suggested above. Therefore:

$$P = \frac{\sum(all \quad probabilities \quad that \quad belong \quad to \quad the \quad winning \quad class)}{(all \quad probabilities \quad considered)}, \tag{23}$$

where winning class is the majority vote, all probabilities: all the probabilities of all classes of all trees

2.6.3 SVM confidence score

SVM is the only ML algorithm that exists in our platform that does not provide any confidence scores. There are studies [28, 31, 39, 43, 46] suggesting different ways to tackle this issue, but the most prominent one is Platt Scaling [31].

Platt Scaling is designed for binary classification, as well as SVMs. The main concept is to pass the SVM scores to a sigmoid function that can result in a confidence score of the SVM score. Therefore:

$$P(y = 1 \mid f) = \frac{1}{1 + e^{(Af+B)}}, \tag{24}$$

where P(y = 1 | f): is the probability of the f(x) SVM output to belong to the positive class, f(x) is the SVM score, A and B are constants by using maximum likelihood from a training set of (fi, yi), where fi SVM score and yi is the true target.

Gradient descent is used to find A and B subject to the solution:

$$argmin_{A,B}[-\sum_i y_i \log(p_i) + (1 - y_i)\log(1 - p_i)] \tag{25}$$

where

$$p_i = \frac{1}{1 + e^{(Af_i+B)}} \tag{26}$$

In order to avoid bias, we split our training set into 90% for model building and 10% for Platt Scaling.

2.6.4 NB confidence score

NBs are known for being good predictors, but bad estimators as they tend to push their prediction probabilities to 0 or 1 [28, 36]. There are many studies that have tried to address this issue [28, 36, 39, 43, 45]. One of the most prominent solutions is Isotonic Regression [46]. Therefore:

$$m' = argmin_z \sum (y_i - z(f_i))^2, \tag{27}$$

where fi: NB score, yi: target class, m: an isotonic (monotonically increasing) function that satisfies:

$$y_i = m(f_i) + \varepsilon_i \tag{28}$$

The solution of Eq.27 could be provided by the pair-adjacent violators (PAV) algorithm [28, 39].

In order to avoid bias, we split our training set into 90% for model building and 10% for Isotonic Regression. Since we support multi-class classification for NB, we had to split the outputs into separate binary ones in order to use Isotonic Regression by creating one model for each class.

The binning method [39, 45] has been considered for NB and found not to be effective enough because there is no ideal method to define the number of bins nor could we uniformly split our probability scores into different bins, since scores were 0 or 1 only.

There is no specific method that could yield the best possible outcome. Platt Scaling is best when the predicted probabilities follow a sigmoid shape, whereas Isotonic Regression caters for any monotonic distortion [28]. Nevertheless, Isotonic Regression is more prone to overfitting for data of small volume than Platt Scaling. Therefore, both methods are considered for the calculation of confidence scores for SVM and NB and this is user-defined.

2.7 Analytics Platform Architecture

The architecture of the analytics platform is shown in Fig. 4

3 Results

We used the 20NewsGroups bydate version that has already split the data into a training set and a test set that is collected in different times to make it more realistic. The data is sorted by date and duplicates as well as some headers are removed. We have separated only 5 of the 20 classes that exist in the dataset, because we observed that they are not so highly related to one another. The classes are atheism,

crypt, baseball, med and space. The training set consists of 2859 documents and the test set of 1900 documents equally split among all classes. Out platform showed very promising results for these classes. The pre-processing method that is used is shown in Table 1 and the evaluation results are shown in Table 2 and Fig. 5. Table 3 shows the best parameters that were found via evaluation for each algorithm and Table 4 shows the time that each algorithm takes for model building and prediction. All tests were done on a computer with Intel Core i7-4720HQ 2.60GHz CPU 32GB RAM.

Many studies [15, 16, 22, 29, 30, 33, 42] have experimented on the publicly available 20NewsGroups dataset. According to these there is not a specific pre-processing technique that can lead to the best evaluation results. Some [22] used the 5000 most frequent words, while others [42] used those words that are not contained in less than 10 documents. Some [15] also kept all words without using stemming and others [33] did not only use all words, but also suggested that feature selection increased error.

In most of these studies SVM and NB are used, as they provide the best results. Their F-measures ranges from 80% - 86% for both algorithms, but this highly de-

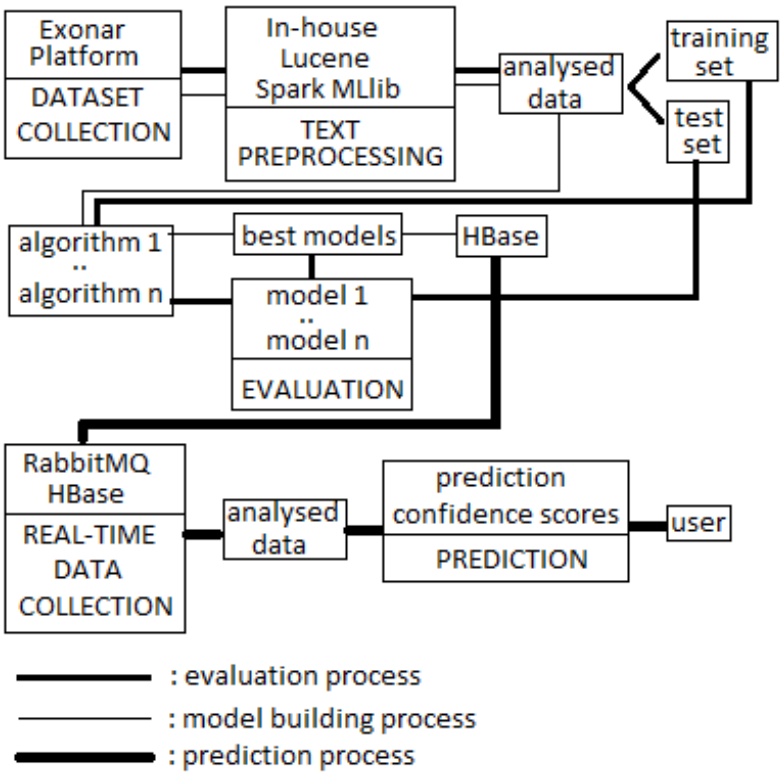


Fig. 4: Platform architecture

Table 1: Pre-processing settings for 20NewsGroups

Keep all features	no
Delete features that are numbers	no
Delete features that are numbers or contain numbers	yes
POS tagging on features	no
Eliminate features that do not exist in $<$ X docs	no
Keep analysed features with length ≥ 2	yes
Keep K most important features in each document	yes, K = 1000
Keep non-analysed features with length ≥ 3	no
Keep all analysed features	no
Feature selection with chi-square	no
Ngrams	no

Table 2: Evaluation results of 20NewsGroups

ALGORITHMS	CLASSES	PRECISION	RECALL	F-MEASURE	TPR	FPR	ACCURACY
DT	atheism	0.9	0.6	0.7	0.6	0	
	baseball	0.2	1	0.4	1	0.86	
	space	0	0	0	0	0	
	crypt	0	0	0	0	0	
	med	0	0	0	0	0	
	AVERAGE	19.79%	30.53%	19.79%	30.53%	18.28%	30.53%
RF	atheism	0.96	0.76	0.85	0.76	0	
	baseball	0.92	0.96	0.94	0.96	0	
	space	0.93	0.92	0.93	0.92	0	
	crypt	0.99	0.88	0.93	0.88	0	
	med	0.74	0.93	0.82	0.93	0.1	
	AVERAGE	90.76%	89.47%	89.63%	89.47%	2.74%	89.47%
SVM	atheism	0.91	0.88	0.9	0.88	0	
	baseball	0.99	0.9	0.94	0.9	0	
	space	0.95	0.91	0.93	0.91	0	
	crypt	0.99	0.93	0.95	0.93	0	
	med	0.96	0.84	0.9	0.8	0	
	AVERAGE	96.08%	89.37%	92.58%	89.37%	0.88%	89.37%
NB	atheism	0.95	0.96	0.95	0.96	0	
	baseball	0.97	0.99	0.98	0.99	0	
	space	0.95	0.97	0.96	0.97	0	
	crypt	0.98	0.97	0.97	0.97	0	
	med	0.97	0.94	0.95	0.94	0	
	AVERAGE	96.54%	96.53%	96.52%	96.53%	0.86%	96.53%

Table 3: Best parameters for each algorithm for classification results in 20News-Groups

ALGORITHM	PARAMETERS	BEST VALUE
DT	TREE DEPTH	10
RF	TREE DEPTH NUMBER OF TREES	29 300
SVM	NUMBER OF ITERATIONS	150
NB	LAMBDA	1

Table 4: Time for model building of 2859 20NewsGroups documents and prediction of 1900 documents

	DT	RF	SVM	NB
Model building time	43.91s	228.61s	51.01s	8.04s
Prediction time	1.62s	2.02s	2.78s	1.97s

pends on the pre-processing methodology that has been used or which version or subset of the 20NewsGroups has been used.

We have observed that keeping all features creates a dictionary of 33763 unique words, deleting numbers gives 29768 words and deleting words containing numbers results in 25643 words in the end. Since all of these options give similar results, we chose the option that creates the smallest vocabulary as shown in Table 1. NB is our best algorithm and we saw a 4% drop in all of its evaluation metrics when

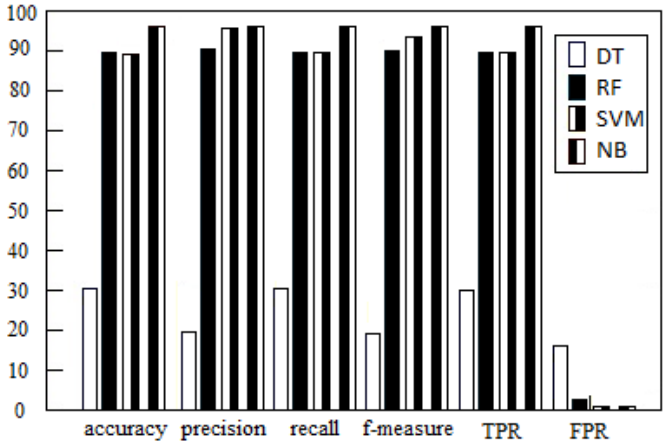


Fig. 5: Best weighted evaluation metrics in 20NewsGroups.

very rare words that did not exist in more than 2 documents are eliminated or when chi-square for K most important words is selected. Keeping all analysed words or keeping analysed words with length > 2 or non-analysed words with length ≥ 3 gives similar results. So, again we chose the one that creates the smallest vocabulary. Keeping K most important words ranging from 200 to 1000 gives the best metrics. On the other hand, when POS tagging or nGrams is used not only they demand more memory and time, but the evaluation metrics drops by 10% for NB as well.

According to Table 2 and Fig. 5 DT behaves very badly, as it can only predict 2 out of 5 classes. The set of different DT parameters that were used did not prove any different in terms of results, but the best one is shown in Table 3. On the other hand, RF behaves very well. It seems that the error induced is due to a trade-off between the recall and precision of atheism and med respectively. The longer the trees the better RF performed. SVM performs very well, but it seems that the precision is higher than the recall. When changing the number of iterations the results were not very different to one another, but the best one was 150. The best algorithm was NB even though it assumes that all words are independent to one another. A wide range of lambdas were used, but the results were almost identical.

NB was not only the most accurate model, but the fastest one as well according to Table 4 followed by DT, SVM and RF. SVM is the most complicating algorithm of all, so we expected it to take longer to be built, but RF is the slowest especially when the number of trees is high. The time of prediction is almost similar for the different algorithms.

The confidence scores during prediction work well for the majority of the predictions, but they can miscalculate a small percentage of the final results.

4 Conclusions

In this project we created a platform for multi-class document classification with the use of Apache Spark. We have supported a number of pre-processing and evaluation techniques on this platform ranging from POS tagging to nGrams, as well as a number of algorithms, like DT, SVM, RF and NB for our model building. We have also supported a real-time prediction process that creates confidence scores for each prediction ranging from methods, like Platt Scaling to Isotonic Regression. Our experiments on the 20NewsGroups dataset showed promising results. As future work, we are going to benchmark these tests on a clustered Spark system for better throughput. We would also like to implement more parallel ML algorithms beginning with K-nearest neighbour as many studies [16, 30] have shown its significance. As a final step, more pre-processing techniques would be evaluated and implemented.

Acknowledgements This work was partly funded by Innovate UK.

References

[1] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, et al (2016) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:160304467

[2] Allen ST, Jankowski M, Pathirana P (2015) Storm Applied: Strategies for real-time event processing. Manning Publications Co.

[3] Baharudin B, Lee LH, Khan K (2010) A review of machine learning algorithms for text-documents classification. Journal of advances in information technology 1(1):4–20

[4] Barga R, Fontama V, Tok WH, Cabrera-Cordon L (2015) Predictive analytics with Microsoft Azure machine learning. Springer

[5] Berthold MR, Cebron N, Dill F, Gabriel TR, Kötter T, Meinl T, Ohl P, Thiel K, Wiswedel B (2009) Knime-the konstanz information miner: version 2.0 and beyond. AcM SIGKDD explorations Newsletter 11(1):26–31

[6] Białecki A, Muir R, Ingersoll G (2012) Apache lucene 4. In: SIGIR 2012 workshop on open source information retrieval, p 17

[7] Borges LC, Marques VM, Bernardino J (2013) Comparison of data mining techniques and tools for data classification. In: Proceedings of the International C* Conference on Computer Science and Software Engineering, ACM, pp 113–116

[8] Breiman L (2001) Random forests. Machine learning 45(1):5–32

[9] Carbone P, Ewen S, Haridi S, Katsifodimos A, Markl V, Tzoumas K (2015) Apache flink: Stream and batch processing in a single engine. Data Engineering p 28

[10] Chapelle O, Vapnik V, Bousquet O, Mukherjee S (2002) Choosing multiple parameters for support vector machines. Machine learning 46(1-3):131–159

[11] Cherkassky V, Ma Y (2004) Practical selection of svm parameters and noise estimation for svm regression. Neural networks 17(1):113–126

[12] De Francisci Morales G (2013) Samoa: A platform for mining big data streams. In: Proceedings of the 22nd International Conference on World Wide Web, ACM, pp 777–778

[13] Demšar J, Curk T, Erjavec A, Gorup Č, Hočevar T, Milutinovič M, Možina M, Polajnar M, Toplak M, Starič A, et al (2013) Orange: data mining toolbox in python. Journal of Machine Learning Research 14(1):2349–2353

[14] George L (2011) HBase: the definitive guide. " O'Reilly Media, Inc."

[15] Guan H, Zhou J, Guo M (2009) A class-feature-centroid classifier for text categorization. In: Proceedings of the 18th international conference on World wide web, ACM, pp 201–210

[16] Guo G, Wang H, Bell D, Bi Y, Greer K (2006) Using knn model for automatic text categorization. Soft Computing 10(5):423–430

[17] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD explorations newsletter 11(1):10–18

[18] Hsu CW, Chang CC, Lin CJ, et al (2003) A practical guide to support vector classification
[19] Kohavi R, et al (1995) A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Ijcai, vol 14, pp 1137–1145
[20] Kottmann J, Margulies B, Ingersoll G, Drost I, Kosin J, Baldridge J, Goetz T, Morton T, Silva W, Autayeu A, et al (????) Apache opennlp. Online (May 2011), www opennlp apache org
[21] Kwon O, Sim JM (2013) Effects of data set features on the performances of classification algorithms. Expert Systems with Applications 40(5):1847–1857
[22] Larochelle H, Bengio Y (2008) Classification using discriminative restricted boltzmann machines. In: Proceedings of the 25th international conference on Machine learning, ACM, pp 536–543
[23] Lee C, Lee GG (2006) Information gain and divergence-based feature selection for machine learning-based text categorization. Information processing & management 42(1):155–165
[24] Liaw A, Wiener M (2002) Classification and regression by randomforest. R news 2(3):18–22
[25] Meng X, Bradley J, Yuvaz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S, et al (2016) Mllib: Machine learning in apache spark. JMLR 17(34):1–7
[26] Mierswa I, Wurst M, Klinkenberg R, Scholz M, Euler T (2006) Yale: Rapid prototyping for complex data mining tasks. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 935–940
[27] Min JH, Lee YC (2005) Bankruptcy prediction using support vector machine with optimal choice of kernel function parameters. Expert systems with applications 28(4):603–614
[28] Niculescu-Mizil A, Caruana R (2005) Predicting good probabilities with supervised learning. In: Proceedings of the 22nd international conference on Machine learning, ACM, pp 625–632
[29] Nigam K, McCallum AK, Thrun S, Mitchell T (2000) Text classification from labeled and unlabeled documents using em. Machine learning 39(2-3):103–134
[30] Pawar PY, Gawande S (2012) A comparative study on different types of approaches to text categorization. International Journal of Machine Learning and Computing 2(4):423
[31] Platt J, et al (1999) Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Advances in large margin classifiers 10(3):61–74
[32] Powers DM (2011) Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation
[33] Rennie JD, Rifkin R (2001) Improving multiclass text classification with the support vector machine

[34] Rennie JD, Shih L, Teevan J, Karger DR, et al (2003) Tackling the poor assumptions of naive bayes text classifiers. In: ICML, Washington DC), vol 3, pp 616–623

[35] Rodriguez JD, Perez A, Lozano JA (2010) Sensitivity analysis of k-fold cross validation in prediction error estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence 32(3):569–575

[36] Schneider KM (2005) Techniques for improving the performance of naive bayes for text classification. In: International Conference on Intelligent Text Processing and Computational Linguistics, Springer, pp 682–693

[37] Sebastiani F (2002) Machine learning in automated text categorization. ACM computing surveys (CSUR) 34(1):1–47

[38] Stahl F, May D, Mills H, Bramer M, Gaber MM (2015) A scalable expressive ensemble learning using random prism: A mapreduce approach. In: Transactions on Large-Scale Data-and Knowledge-Centered Systems XX, Springer, pp 90–107

[39] Takahashi K, Takamura H, Okumura M (2009) Direct estimation of class membership probabilities for multiclass classification using multiple scores. Knowledge and information systems 19(2):185–210

[40] Wahbeh AH, Al-Radaideh QA, Al-Kabi MN, Al-Shawakfa EM (2011) A comparison study between data mining tools over some classification methods. IJACSA) International Journal of Advanced Computer Science and Applications, Special Issue on Artificial Intelligence pp 18–26

[41] Wang X, McCallum A, Wei X (2007) Topical n-grams: Phrase and topic discovery, with an application to information retrieval. In: Seventh IEEE International Conference on Data Mining (ICDM 2007), IEEE, pp 697–702

[42] Wu M, Schölkopf B (2006) A local learning approach for clustering. In: Advances in neural information processing systems, pp 1529–1536

[43] Wu TF, Lin CJ, Weng RC (2004) Probability estimates for multi-class classification by pairwise coupling. Journal of Machine Learning Research 5(Aug):975–1005

[44] Yang Y, Pedersen JO (1997) A comparative study on feature selection in text categorization. In: ICML, vol 97, pp 412–420

[45] Zadrozny B, Elkan C (2001) Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In: ICML, Citeseer, vol 1, pp 609–616

[46] Zadrozny B, Elkan C (2002) Transforming classifier scores into accurate multiclass probability estimates. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 694–699

[47] Zuber M (2014) A survey of data mining techniques for social network analysis. International Journal of Research in Computer Engineering & Electronics 3(6)