# A Text Mining Framework for Big Data

Niki Pavlopoulou, Aeham Abushwashi, Frederic Stahl and Vittorio Scibetta

**Abstract** Text Mining is the ability to generate knowledge (insight) from text. This is a challenging task, especially when the target text databases are very large. 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 are required to process a large amount of data. However, there is no open-source end-to-end framework that can combine near real-time and batch processing of ingested big textual data along with user-defined options and provision of specific, reliable insight from the data. This is important as this way new unstructured information is made accessible in near real-time, more personalised customer products can be created and novel unusual patterns can be found and actioned on quickly. This work focuses on a proprietary complete near real-time automated classification framework for unstructured data with the use of Natural Language Processing and Machine Learning algorithms on Apache Spark. The evaluation of our framework shows that it achieves a comparable accuracy with respect to some of the best approaches presented in the literature.

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

Frederic Stahl
Department of Computer Science, University of Reading, Whiteknights, Reading, Berkshire, RG6 6UA, UK, e-mail: f.t.stahl@reading.ac.uk

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

## 1 Introduction

The automatic classification of unknown text documents is an important problem in data mining. The amount of text available for business analytics is vast, originating out of sources ranging from the World Wide Web, social media, e-mails, medical records, databases etc. Much of this data inherently lacks coherent structure. There is far too much data for human users to manually process and categorise. Therefore, methods like Text Mining have emerged to solve this problem [2].

Classification in Text Mining is the ability to automatically attach a label to ~~a~~ previously unseen documents using models extracted by algorithms from a collection of known and labelled documents [40]. The acquisition of such a collection of texts that can be used to train such a model is a challenging task on its own. Most of the time such a training set is created manually by subject matter experts who can identify documents that represent each label best. This process is often biased by the expert and time-consuming. Furthermore, it is often unclear how many text documents are needed to generate an accurate predictive model. Often the 'rule of thumb' is 'the more the better'. However, this is not true for all cases [22].

After the training set is gathered, a number of pre-processing techniques are used [4]. 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 this selection process.

Once the input data has been pre-processed, Machine Learning (ML) algorithms are used in order to build models that can learn from the training set and can later be used to classify previously unseen documents [4]. There is a plethora of algorithms to do this and one can determine through experimentation which ones work best for a specific problem.

There are a number of frameworks and tools for Data Mining [8, 43], but most are not open-source. RapidMiner [27], Orange [14], KNIME [6], Weka [18] are on the other hand some of the most popular open-source tools that support a myriad of algorithms, pre-processing methods and rich GUIs. However, they typically do not cope with very large amounts of data nor do most of them have support for stream processing. Whilst these tools and frameworks are perfectly usable in environments where the 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 given to streaming Big Data frameworks such as Apache Spark [26], Apache Storm [3] and Apache Flink [10]. Others, like Azure ML [5], SAMOA [13] and TensorFlow [1] are also very powerful tools. However, Azure ML supports cloud services only and thus blocks out users who need the cloud computing capability on premise. SAMOA, on the other hand, supports only streaming data, which is not applicable for all applications. Also in our experience Deep Learning frameworks, like TensorFlow are frequently used for industrial problems where in fact traditional ML techniques would be sufficient.

We focused our attention on Apache Spark, which is widely used in production by many organisations and 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 variety 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 and be able to 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/framework, Section 3 provides results using the *20NewsGroups* dataset and Section 4 describes conclusions and future work.

## 2 Methods

Our platform consists of two processes: the model building and the near real-time prediction of document labels. In order for these processes to take place, a number of stages need to be supported. These stages are described in this Section.

### *2.1 Data Collection*

The first step in model building is the creation of the training set. In our case Exonar's search and discovery platform (https://www.exonar.com/platform/) creates the training set and it can span from publicly available data to client data. This platform collects and analyses enterprise data, stores it in a NoSQL database (HBase) [15], and enables users to easily identify training sets for a given classification task.

### *2.2 Text Pre-processing*

Raw text is not suitable as direct input to a classification engine. The text needs to be represented in a way that is suitable as input for ML algorithms. There are a number of techniques that can be used [40] for this purpose. Below are some of the most prominent ones that are implemented in our platform and the user has the ability to choose from these techniques. The implementation is done with a combination of in-house methods, Spark, Apache Lucene [7] and OpenNLP [21].

#### 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 memory requirements and it is used 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 particular weighting scheme. Feature extraction techniques are then used to analyse the original words and results in a concise and more representative dictionary.

#### 2.2.2 Feature Extraction

Texts in general may contain lots of ambiguity, syntactical errors, frequent 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 semantics, filters are used and their results are shown in Fig. 1.

Tokenisation: transforms a document $d_i$ to a collection of words ($w_1$, $w_1$, .., $w_n$) where $n$ is the number of unique words in $d_i$, often called ‘bag of words’. Simple tokenisers, like the whitespace tokeniser illustrated in Fig. 1, are used in addition to some more specialised tokenisers that can capture specific types of entities, such as URLs.

Lowercase filter: transforms all words into lowercase characters. Thus words such as ‘Runner’ or ‘runner’ can be identified as the one token ‘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’. In our framework the Porter Stemmer [33] for English was used was used, in particular Lucene’s implementation of the stemmer.

#### 2.2.3 Feature Selection

One of the main challenges in Text Mining is high-dimensionality. Texts often contain a lot of words that can range from very important to highly noisy. Different data sets can have distinctly different characteristics. It is useful for 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. However, that would consume a lot of memory and the ML algorithms will require more time to process the data. Another option is to discard words (before stemming) that have less than 4 characters as they are less likely to be significant compared with longer words.

Another option could be to keep the $k$ most important words, where $k$ is user-defined using some measure to quantify importance.

Words that are extremely rare throughout the collection $D$ may be noise. Our system has an option for eliminating those words that exist in less than **x** documents, where $x$ is user-defined. This is expected to result in removing rare words and possibly a higher accuracy.

Information gain [24] is a statistical method used by Decision Tree (DT) and Random Forest (RF) for selecting the most significant features throughout the dictionary. A sample $s_i$ of the collection $D$ is related to documents that belong to a specific class $c_i$. A word $w_i$ may be contained in this sample or not. Entropy $H$ quantifies how homogeneous $s_i$ is. The lower the entropy the higher the homogeneity. Therefore, the information gain of a word $w_i$ for a sample $s_i$ is:

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

where $p(w_i)$ is the probability of $w_i$ occurring, p($w_i'$) is the probability of $w_i$ not occurring, p($c_i$) is the probability of the $i^{th}$ class, p($c_i \mid w_i$) is the probability of the $i^{th}$ class given the occurrence of $w_i$ and p($c_i \mid w_i'$) is the probability of the $i^{th}$ class

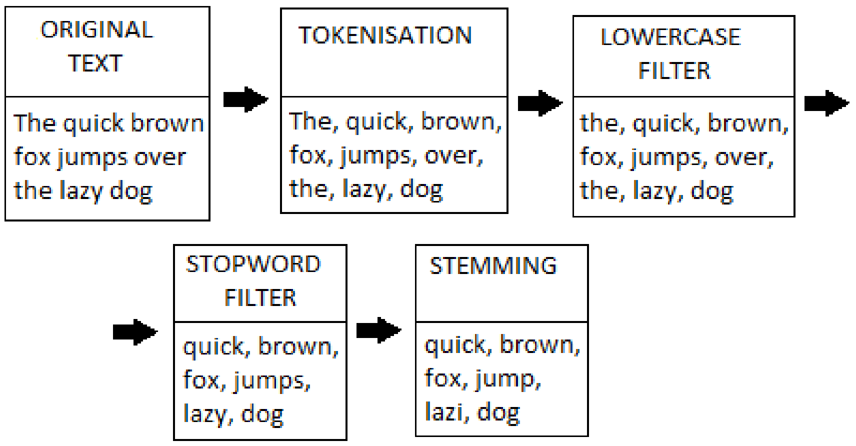

ORIGINAL TEXT
The quick brown fox jumps over the lazy dog
TOKENISATION
The, quick, brown, fox, jumps, over, the, lazy, dog
LOWERCASE FILTER
the, quick, brown, fox, jumps, over, the, lazy, dog
STOPWORD FILTER
quick, brown, fox, jumps, lazy, dog
STEMMING
quick, brown, fox, jump, lazi, dog

Fig. 1: Stages of feature extraction from text

given the absence of $w_i$. As the algorithm traverses all possible words, the ones that have the highest information gain will be selected in the end.

Chi-square [47] is a statistical method that examines the dependency between a word $w_i$ and a class $c_i$. It is defined as:

$$\chi^2(w_i,c_i) = \frac{|D|(N_{w_ic_i}N_{w'_ic'_i} - N_{w'_ic_i}N_{w_ic'_i})^2}{(N_{w_ic_i}+N_{w'_ic_i})(N_{w_ic'_i}+N_{w'_ic'_i})(N_{w_ic_i}+N_{w_ic'_i})(N_{w'_ic_i}+N_{w'_ic'_i})} \quad (2)$$

$$\chi^2_{avg}(w_i) = \sum_{i=1}^{m} P(c_i)\chi^2(w_i,c_i) \quad (3)$$

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

where |D| is the number of documents in the collection $D$, $N_{w_ic_i}$ is the number of times $w_i$ occurs in documents belonging to class $c_i$, $N_{w_ic'_i}$ is the number of times $w_i$ occurs in documents that do not belong to class $c_i$, $N_{w'_ic_i}$ is the number of times $c_i$ occurs without the word $w_i$ and $N_{w'_ic'_i}$ is the number of times neither $c_i$ nor $w_i$ occurs. The higher the chi-square the more dependent $w_i$ and $c_i$ are. In our system a selection of the $k$ most dependent words for each class are selected, where $k$ is user-defined.

### 2.2.4 Feature Representation

A frequently used feature representation model is the bag-of-words or n-grams. In the bag-of-words the dictionary consists of single words. However, sometimes the meaning of words, the sequence of them and the phrases that exist play a vital role [44]. Providing only individual words to the ML algorithm may 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 the best results. The user can either use unigrams or n-grams of n $>$ 1, where $n$ is user-defined.

When using n-grams it is sometimes better 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 the bi-gram of ‘state art’, which has a different meaning. Stemming could also alter the original meaning of phrases. However, n-grams are memory-intensive compared with unigrams as more words are created. Also the stopword removal and stemming can be useful to improve the time performance of the ML algorithms. The choice of using stopword removal or stemming for n-grams 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 former context and a noun in the latter context. 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 of our system has the choice whether to use tagging or not.

### 2.2.5 Weighting Scheme

As explained earlier, each document is represented by a vector with indices representing the mapped words and values of their corresponding weight. This weight derives from a choice of different weighting schemes with the most popular one being the Term Frequency - Inverse Document Frequency (TF-IDF) metric.

TF-IDF captures the uniqueness and significance of each word 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, for our system we have experimented with 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 generally acceptable for some of our cases and TF-IDF seemed to work well in almost all cases. The TF-IDF is calculated as:

$$TF = 1 + \log(f_{w_i,d_i}) \tag{5}$$

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

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

where $f_{w_i,d_i}$ is the frequency of the word $w_i$ in the document $d_i$, |D| is the number of documents and DF($w_i$, $D$) is the number of documents that contain the word $w_i$.

The concept behind this metric is 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 as it 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 [35] and RF [9] 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 that the dataset is balanced for all classes. If the dataset contains too many documents of one class and only a few of another, then a model is bound to be biased towards the class with the most documents, hence, overfitting of the model is likely. In our platform, if there is sufficient data for all classes, then balancing is done automatically through a user specified number of documents per class.

## *2.3 Evaluation*

In our platform, 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 a configuration file. In general, during evaluation, the training set $D$ is split into a subset of the training set $D_{tr}$ and a test set $D_{te}$. $D_{tr}$ is used to build the model and $D_{te}$ 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 starts an evaluation, the accuracies for each class are calculated and averaged and all evaluation metrics are stored. The implementation has been done from scratch in Spark.

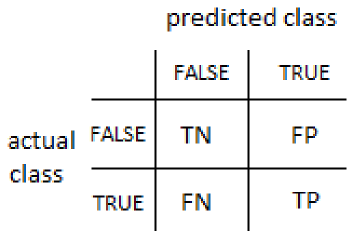
The evaluation methods are described in Sections 2.3.1 to 2.3.3.

### 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 first round the first fold is used as $D_{te}$ and the rest as $D_{tr}$, in the second round the second fold is used as $D_{te}$ and the rest as $D_{tr}$ and so on until each fold has been used as $D_{te}$ exactly once. The final evaluation metric consists of the average value of the metrics of all rounds. This is considered to be a robust 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 a good setting for cross validation [38].

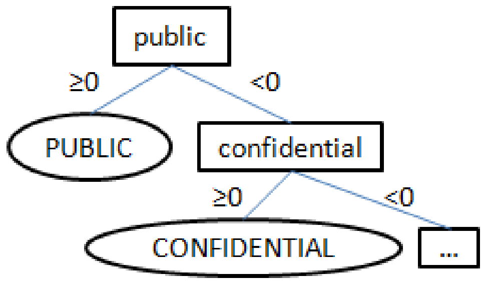
### 2.3.2 Holdout

This method splits $D$ into 70% as $D_{tr}$ and 30% as $D_{te}$ (using random sampling without replacement). The method is not considered to be as robust 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 [20].



|  |  | predicted class |  |
|---|---|---|---|
|  |  | FALSE | TRUE |
| actual class | FALSE | TN | FP |
|  | TRUE | FN | TP |

(a) Confusion matrix



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

Fig. 2: Confusion Matrix and a DT

### 2.3.3 Evaluation Metrics

Since $D_{te}$ contains the real classes, one is able to estimate how well the model behaves by comparing the true class label versus the predicted ones. A confusion matrix [34] as shown in Fig. 2a is useful for binary predictions where the class label is either positive or negative and it contains all True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN).

There are more specific evaluation metrics that can be used to cater for the model's performance using the TP, TN, FN and FP values from the confusion matrix. The following more specific evaluation metrics are implemented in our platform:

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

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.

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

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.

$$FalsePositiveRate(FPR) = fall-out = \frac{FP}{TN+FP} \tag{12}$$

FPR is the proportion of negatives that are predicted as positives

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

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

## *2.4 Machine Learning Algorithms*

Once the evaluation process is done, the platform picks 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 Naïve Bayes (NB) that are available on Spark.

### 2.4.1 Decision Trees

A DT [35] 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 (according to some metric such as information gain) followed by the less significant features until a user-defined depth is reached or no further branches are possible. 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 Random Forests

A RF [9, 25, 41] 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 most frequently predicted class among all the trees the document's features have traversed. The advantage of RF is that it is less prone to overfitting compared with DTs, but it is has a much higher computational cost 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 Support Vector Machines

SVM [19, 28] 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 the closest to the MMH are called support vectors.

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). A new document collects its predictions from all models and the one that responds to a positive one will win.

One of the main advantages is that SVM can reach a global optimum and is less prone to overfitting. It has been shown that it can handle high-dimensional data [4]. However, one of the main issues is the computational cost, especially for non-linear SVM and the parameter selection of non-linear SVMs [11, 12].

### 2.4.4 Naïve Bayes

NB is a probabilistic model that is based on Bayes' Theorem. It is called naïve , 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 [37] version is used. The likelihood of a document is defined as:

$$p(d_i \mid \theta_c') = \frac{(\sum_i f_{w_i,d_i})!}{\prod_i f_{w_i,d_i}!} \prod_i (\theta_{ci})^{f_{w_i,d_i}}, \tag{13}$$

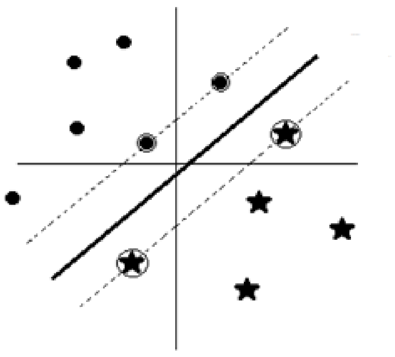


**Fig. 3** An SVM for a binary classification of stars versus circles that are linearly separable. The thick black line is the optimal hyperplane and the circled points on the dashed line are the support vectors for circles and stars respectively

where $c$ is a class, $\theta'_c$ is the vector of a class $c$ with values of $\theta_{ci}$ that is the probability that word $w_i$ occurs together with class $c_i$; and $f_{w_i,d_i}$ is the frequency count of word $w_i$ in document $d_i$.

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

$$\begin{aligned} l(d) &= argmax_c[\log p(\theta'_c) + \sum_i f_{w_i,d_i} \log \theta_{ci}] \\ &= argmax_c[b_c + \sum_i f_{w_i,d_i} W_{ci}], \end{aligned} \tag{14}$$

where $b_c$ is the threshold term and $W_{ci}$ is the weight of class $c_i$ for word $w_i$.

The user can select the smoothing parameter in $\theta$, 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 0.

## 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 may be obsolete. Furthermore, streaming could also be used for real-time user demands to build different (and new) models. 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 near real-time in order to cater for these demands.

## 2.6 *Near real-time Prediction*

After the model building phase (training) 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 new documents are collected and the same pre-processing procedure as the one applied during the model generation process is executed. 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 reliable the prediction is. This whole process was implemented from scratch in Apache Spark.

The confidence scores are used in our system to establish confidence in the quality of predictions. We observed that some ML algorithms can be well-calibrated, that is the prediction probability matches the confidence score closely, but there are others that perform poorly in this regard. Other algorithms do not support extraction of confidence scores at all. There are many studies [29, 32, 39, 42, 46, 48, 49] that aimed to address this issue.

### 2.6.1 Decision Tree 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:

$$P = \frac{k}{n}, \tag{15}$$

where $k$ is the number of training instances under that leaf that matches the leaf's class and $n$ is the total number of training instances under the leaf.

The authors of [48] suggest that this score is biased since DTs tend to create homogeneous leaves or are statistically unreliable when the instances that fall under a leaf are small in numbers.

### 2.6.2 Random Forests Confidence Score

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

$$P = \frac{\sum(confidence \quad scores \quad of \quad trees \quad voting \quad for \quad the \quad winning \quad class \quad)}{\sum(confidence \quad score \quad of \quad all \quad trees)} \tag{16}$$

where the 'winning class' is the majority vote and 'all probabilities'are 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 [29, 32, 42, 46, 49] suggesting different ways to tackle this issue, but the most prominent one is Platt Scaling [32].

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:

$$P(y=1 \mid f) = \frac{1}{1+e^{(Af+B)}}, \tag{17}$$

where $P(y=1 \mid f)$ is the probability of the SVM output to belong to the positive class, $f$ is the SVM score. A and B are constants by using maximum likelihood from a training set of $(f_i, y_i)$, where $f_i$ is the SVM score of a document $d_i$ and $y_i$ is its true target. Gradient descent is used to find A and B subject to the solution:

$$argmin_{A,B}[-\sum_i y_i \log(p_i) + (1-y_i)\log(1-p_i)] \tag{18}$$

where

$$p_i = \frac{1}{1+e^{(Af_i+B)}} \tag{19}$$

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 towards 0 or 1 [29, 39]. There are many studies that have tried to address this issue [29, 39, 42, 46, 48]. One of the most prominent solutions is Isotonic Regression [49]. Therefore:

$$m' = argmin_z \sum (y_i - z(f_i))^2, \tag{20}$$

where $f_i$ is the NB score, $y_i$ is the target class, $m$ is an isotonic (monotonically increasing) function that satisfies:

$$y_i = m(f_i) + \varepsilon_i \tag{21}$$

The solution of Eq.20 can be provided by the Pair-Adjacent Violators (PAV) algorithm [29, 42].

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 split the outputs into separate binary ones in order to use Isotonic Regression by creating one model for each class.

The binning method [42, 48] has been considered for NB in our system 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 our scores were pushed close to either 0 or 1.

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 [29]. 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 can be defined by the user in the developed system.

### *2.7 Analytics Platform Architecture*

The architecture of the analytics platform is shown in Fig. 4. Our architecture consists of three stages: the model building process, the evaluation process and the prediction process. During the model building process, a dataset that is relevant to the classification task is collected by the Exonar platform and then the data is then pre-processed with in-house, Lucene or Spark MLlib methods. Then the analysed (pre-processed) data is used to create different (predictive) models that are in turn stored into HBase for future use. If evaluation does not take place, then the default parameters in the configuration file, both for pre-processing and algorithms are taken into consideration. On the other hand, if evaluation takes place, then the analysed data is split into two separate datasets, the training set and the test set. The training set is then used to create the models and the test set is used to find the best models. The set of different parameters for the evaluation process, as well as whether the evaluation will be done with cross-validation or random-split are specified in the configuration file. The best models are then stored into HBase for future use. The last process is the prediction. In this process the data (document) IDs that need to be categorised are pushed into RabbitMQ and their contents is then retrieved from HBase. The same pre-processing techniques as in the model building process are used to extract the most important information from these documents and then the predicted class and the confidence score for the class is calculated by retrieving specific models from HBase that again are defined in the configuration file. Then the result of the predicted class and its confidence score for specific data (document) IDs are presented through the Exonar platform's user interface.

## 3 Results

We used the 20NewsGroups 'by date' version that has already been split into a training set and a test set. This is more realistic because the documents in the training set are older than the documents in the test set. The data is sorted by date and duplicates and some headers are removed. We have separated 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. Our platform showed very promising results for these classes. The pre-processing methods used are 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.

Table 1: Pre-processing settings for 20NewsGroups

| PRE-PROCESSING TECHNIQUE | SETTING |
|---|---|
| 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 $\geq 2$ | yes |
| Keep K most important features in each document | yes, K = 1000 |
| Keep non-analysed features with length $\geq 3$ | no |
| Keep all analysed features | no |
| Feature selection with chi-square | no |
| n-grams | no |

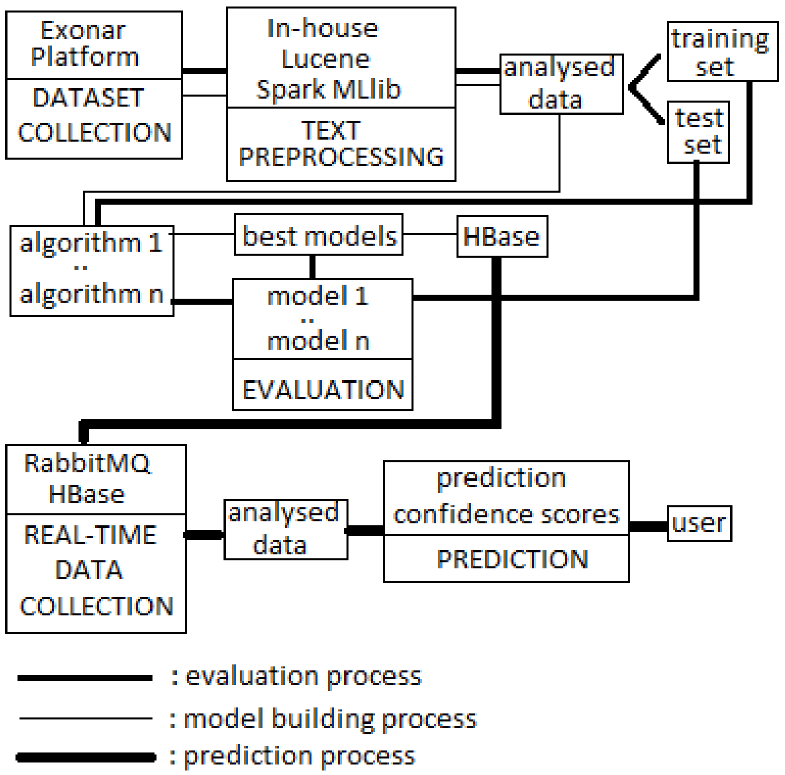


Fig. 4: Platform architecture

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 20NewsGroups

| ALGORITHM | PARAMETERS | BEST VALUE |
|---|---|---|
| DT | TREE DEPTH | 10 |
| RF | TREE DEPTH | 29 |
| | NUMBER OF TREES | 300 |
| SVM | NUMBER OF ITERATIONS | 150 |
| NB | SMOOTHING PARAMETER | 1 |

Many studies [16, 17, 23, 30, 31, 36, 45] have experimented on the publicly available 20NewsGroups dataset. According to these there is no specific pre-processing technique that can lead to the best evaluation results. Some [23] used the 5000 most frequent words, while others [45] used those words that are not contained in less than 10 documents. Some [16] also kept all words without using stemming and others [36] did not only use all words, but also suggested that feature selection increased error.

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 |

In most of these studies SVM and NB were used, as they provide the best results. Their F-measure ranges from 80% - 86% for both algorithms, but this depends 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. 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 see a 4% drop in all of its evaluation metrics when very rare words that do 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 $\geq 3$ gives similar results. So we chose the one that creates the smallest vocabulary (analysed words with length $\geq 2$). Keeping *K* most important words ranging from 200 to 1000 gives the best results. On the other hand, when POS tagging or n-grams is used not only they demand more memory and time, but the evaluation metrics drops by 10% for NB as well.

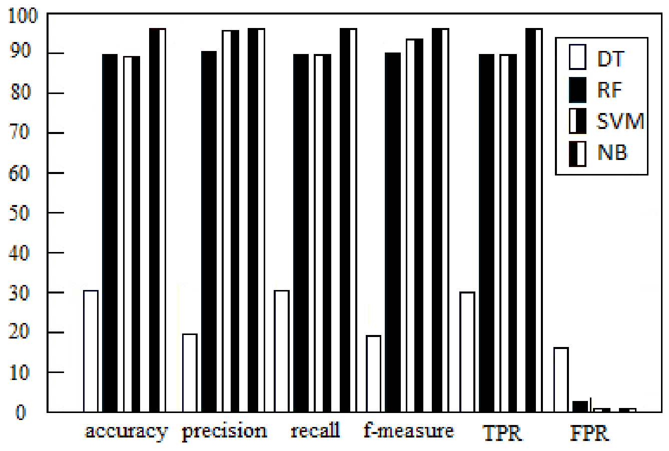


Fig. 5: Best weighted evaluation metrics in 20NewsGroups

According to Table 2 and Fig. 5 DT achieves very low metrics and an analysis of its predictions confirms that DT is only able to predict 2 out of 5 classes well. The set of different DT parameters that are used do 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. An analysis of its predictions shows that its only significant error is a result of a trade-off between the recall and precision of atheism and med classes respectively. Also we observe the deeper the trees the better RF performs. SVM performs very well, but their precision is higher than their recall. When changing the number of iterations the results are not very different to one another, but the best results are obtained for 150 iterations. The best algorithm is NB even though it assumes that all words are independent to one another. A wide range of smoothing parameters are used, but the results are almost identical.

NB is not only the most accurate model, but the fastest one as well according to Table 4 followed by DT, SVM and RF. The time of prediction is almost the same 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 n-grams, as well as a number of algorithms, like DT, SVM, RF and NB for building a classification model. We have also supported a near 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 also plan to implement more parallel ML algorithms beginning with K-Nearest Neighbour as many studies [17, 31] have shown its significance. Finally, more pre-processing techniques will be implemented and evaluated.

**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] Adedoyin-Olowe M, Gaber MM, Stahl F (2014) A survey of data mining techniques for social media analysis. Journal of Data Mining & Digital Humanities 2014

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

[4] 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

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

[6] 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

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

[8] 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

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

[10] 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

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

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

[13] 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

[14] 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

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

[16] 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

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

[18] 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

[19] Hsu CW, Chang CC, Lin CJ, et al (2003) A practical guide to support vector classification

[20] 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

[21] Kottmann J, Margulies B, Ingersoll G, Drost I, Kosin J, Baldridge J, Goetz T, Morton Tea (Accessed: February 2017) Apache OpenNLP. www. opennlp. apache. org

[22] Kwon O, Sim JM (2013) Effects of data set features on the performances of classification algorithms. Expert Systems with Applications 40(5):1847–1857

[23] 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

[24] 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

[25] Liaw A, Wiener M (2002) Classification and regression by randomforest. R news 2(3):18–22

[26] 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

[27] 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

[28] 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

[29] 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

[30] 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

[31] 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

[32] 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

[33] Porter MF (1997) Readings in information retrieval. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, chap An Algorithm for Suffix Stripping, pp 313–316, URL http://dl.acm.org/citation.cfm?id=275537.275705

[34] Powers DM (2011) Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation

[35] Quinlan JR (1986) Induction of decision trees. Mach Learn 1(1):81–106, DOI 10.1023/A:1022643204877, URL http://dx.doi.org/10.1023/A:1022643204877

[36] Rennie JD, Rifkin R (2001) Improving multiclass text classification with the support vector machine

[37] 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

[38] 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

[39] 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

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

[41] 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

[42] 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

[43] 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

[44] 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

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

[46] 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

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

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

[49] 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