



A Parallel Algorithm for Bayesian Text Classification Based on Noise Elimination and Dimension Reduction in Spark Computing Environment

Zhuo Tang¹(✉), Wei Xiao¹, Bin Lu¹, Youfei Zuo¹, Yuan Zhou², and Keqin Li¹

¹ College of Information Science and Engineering, Hunan University,
Changsha 410082, Hunana, China
ztang@hnu.edu.cn

² College of Public Policy and Management, Tsinghua University,
Beijing 100084, China

Abstract. The Naive Bayesian algorithm is one of the ten classical algorithms in data mining, which is widely used as the basic theory for text classification. With the high-speed development of the Internet and information systems, huge amount of data are being produced all the time. Some problems are certain to arise when the traditional Bayesian classification algorithm addresses massive amount of data, especially without the parallel computing framework. This paper proposes an improved Bayesian algorithm INBCS, for text classification in the Spark computing environment and improves the Naive Bayesian algorithm based on a polynomial model. For the data preprocessing, this paper first proposes a parallel noise elimination algorithm, and then proposes another parallel dimension reduction algorithm based on Information Gain and TextRank computation in the Spark environment. Based on these preprocessed data, an improved parallel method is proposed for calculating the conditional probability that comprehensively considers the effects of the feature items in each document, class and training set. Finally, through experiments on different widely used corpuses on the Spark computation platform, the results illustrate that INBCS can obtain higher accuracy and efficiency than some current improvements and implementations of the Naive Bayesian algorithms in Spark ML-library.

Keywords: Big data · Naive bayesian classification · Parallel computing · Spark · Text classification

1 Introduction

With the rapid development of information society, the Internet has been widely used and currently has become the most important source of information. In particular, with the emergence of cloud computing and the big data era [1],

the data generated from the Internet are rapidly growing with the index grade. These data have the following characteristics: large in amount, high in dimension, complex in structure and containing much noise, but widespread application prospects. Furthermore, most of the information and data stored on the Internet are text. How to organize, manage, and utilize these text data is a great challenge for the currently limited computing power, especially when confronts with a large amount of information that needs to be searched effectively, quickly and accurately by users for the Internet applications. The Spark platform [2], as the new generation of big data processing engine, supports a Resilient Distributed Datasets (RDD) model built on a in-memory computing framework. It allows users to cache data in memory, and to perform computation and iteration for the same data directly from memory. Based on the memory computing mode, the Spark platform can save the amounts of disk I/O operation time. Therefore, it is more suitable for machine learning algorithms with iterative computation.

This paper proposes an improved naive Bayesian classifier model based on the Spark platform (INBCS), which provides a new method to calculate the conditional probability and solve the above problems. INBCS uses TF-IDF weighting to obtain the probabilities of feature items belonging to a given class. It not only takes account of the respective proportions in feature items and the entire training set but also takes the impact of the proportion of documents into consideration, which contains the feature items in the training set. From the above, our contributions in this paper are summarized as follows:

1. A parallel method that removes noises in a data set running on the Spark platform.
2. A dimension reduction method for high-dimensional data in English and Chinese texts preprocessing.
3. An improved conditional probability applied in Naive Bayesian to improve the precision and accuracy.
4. A new memory prediction algorithm used in the *SpillWrite* operation of Spark's *Shuffle Read*.

The rest of the paper is organized as follows. Section 2 reviews the background and related work. Section 3 proposes an improved Multinomial Bayesian model, INBCS, and provides the formalization descriptions for this model. Parallel implementation of INBCS on Spark is developed in Sect. 4. Experimental results and evaluations are provided in Sect. 5. Finally, Sect. 6 presents the conclusion and future works.

2 Background and Related Work

Several machine learning and data mining algorithms have been proposed for text classification. The most popular methods include Naive Bayesian [3], support vector machines [4], decision trees, artificial neural networks, k -nearest-neighbor (k -NN) classification and association rules. The support vector machine model

has been proven to be more accurate than most other techniques for classification, but the complexity of the algorithm is relatively high [5]. Compared to other algorithms, a decision tree is simple and easy to understand. However, its accuracy is unsatisfactory compared to other text classification algorithms, especially when the number of distinguishing features of documents becomes large. The k -NN algorithm is easy to implement and shows its effectiveness in a variety of problem domains. However, the computation will increase dramatically when the size of the training set grows large. Naive Bayes is based on an independence assumption for the features in a document. Although this assumption violates the natural language rules, after IR transformation [6], the results show that Naive Bayes classification can still perform surprisingly well [3]. Because Naive Bayes is efficient and easy to implement, this paper proposes an improved model INBCS for the text classification and improves its performance through noise diminution and dimension reduction, especially for various practical applications, such as spam filtering or news article classification.

Feature selection and reduction are extremely important phases in classification algorithms, and there have been multiple efforts to improve them in a variety of scenarios. Aghdam et al. [7] introduced a feature selection and reduction method using ant colony optimization. Shi et al. [8] considered test criteria, such as frequency, dispersion and concentration indices, and proposed an improved dimension reduction method and feature weighting method to make the selection more representative and the weighting of characteristic features more reasonable. Berka et al. [9] made another effort for the improvement of dimensionality reduction and introduced an algorithm that replaces rare terms by computing a vector that expresses their semantics in terms of common terms. Furthermore, due to the high dimensionality of data, Xu [10] proposed a dimension reduction method for the registration of high-dimensional data. Tao et al. [11] and Lin et al. [12] analyzed and proposed various improved classification algorithms for high-dimensional data based on dimension reduction. However, all the serial algorithms above achieved improved performance on small-scale data. When the scale and dimension of data become large, the effects of these algorithms would be reduced.

How to eliminate the noise in the text has always been a topic that must be considered in the text classification algorithm, too much noise in the text will not only increase the calculation amount of the processing, but also seriously affect the precision and accuracy of the classification. The researchers have made great progress in this area. For example, *RONG-LU LI* mentioned a method that based on text density and sample distribution can be used to eliminate noise data in the training data set, at the International Conference on Machine Learning and Cybernetics conference in 2003. In addition, how to classify text more quickly and effectively has also bothered people, so the concept of parallel and distributed computing has gradually become the focus of public attention. There are also many related researches on the implementation of text classification algorithms based on parallel computing platforms, such as, *Xiangxiang Chen* and *Kaigui Wu*'s paper presented at the 2010 MINS conference, which

proposes a parallel distributed classification algorithm based on Mapreduce model to reduce computational time during large numbers of training process. At same time, *skender lgen Oul* introduced a fast Bayesian text classification method on the spark distributed platform, because the spark platform uses a distributed in-memory data structure to provide faster storage and analysis of data. Based on the previous research, this paper proposes a parallel algorithm for bayesian text classification based on noise elimination and dimension reduction in spark computing environment.

3 An Improved Multinomial Bayesian Model

3.1 The Naive Bayes Classifiers

The Bayes theorem describes the probability of an event, which is based on conditions that might be related to previous events. Naive Bayes classifiers are based on strong independence assumptions among all features in the samples. We define C as a set of predefined categories, which consists of m components as follows: $C = \{c_1, c_2, \dots, c_m\}$. This paper defines $P(c_j|d_i)$ to denote the probability of a document d_i belonging to a class c_j . The classifier selects the class with the maximum probability as the result class, and the conditional probability $P(c_j|d_i)$ can be calculated by the Bayesian theorem as in Eq. (1):

$$P(c_j|d_i) = \frac{P(c_j)P(d_i|c_j)}{P(d_i)} \quad (1)$$

where $P(d_i|c_j)$ represents the distribution of documents in each class, and it cannot be estimated directly. We use this formula for text classification specifically. However, based on the *Naive Bayes Assumption*, the document d_i can be treated as a sequence SEQ_i with a set of independent words w_k . Therefore, the length of document d_i can also be regarded as the number of words in SEQ_i , and $P(d_i|c_j)$ can be calculated by Eq. (2):

$$P(d_i|c_j) = \prod_{k=1}^n P(w_k|c_j) \quad (2)$$

In Eq. (1), $P(d_i)$ represents the probability of a document d_i in the training data set. Because the values of $P(d_i)$ are the same for all the classes, we can ignore this probability when we compare $P(c_j|d_i)$ to other classes.

Because a Bayesian text classifier formalizes the distribution of words in a document as a multinomial, a document can be formalized as a sequence of words with the assumption that each word position is generated independently. Based on this, we assume that there are a fixed number of classes: $C = \{c_1, c_2, \dots, c_m\}$ and that each of them has a fixed set of multinomial parameters. For a specific class c , the parameter vector is $\vec{V}_c = \{V_{c,1}, V_{c,2}, \dots, V_{c,n}\}$, where n is the vocabulary size, $\sum_j V_{c,j} = 1$, and $V_{c,j}$ is the probability that word j appears in

class c [6]. The likelihood of a document depends on the parameters of the words that appear in the document:

$$P(d|\vec{V}_c) = \frac{(\sum_j f_j)!}{\prod_j f_j!} \prod_j (V_{c_j})^{f_j} \quad (3)$$

where f_j denotes the frequency count of each different word in the document d . By assigning $P(\vec{V}_c)$ as a prior distribution over the set of classes, we can acquire the minimum-error classification rule that selects the class with the largest posterior probability, as in Eq. (4):

$$L(d) = \arg \max_c \left[\log P(\vec{V}_c) + \sum_j f_j \log V_{c,j} \right] = \arg \max_c \left[h_c + \sum_j f_j W_{c,j} \right] \quad (4)$$

where h_c is the threshold term and $W_{c,j}$ denotes the weight of *class* $_c$ for *word* $_j$.

3.2 An Improved Naive Bayesian Text Classifier

To process the documents more efficiently, this section uses a Vector Space Model to represent the texts, which is often used in information filtering, information retrieval, indexing, and relevancy rankings. In INBCS, we formalize a set of documents as $\vec{d} = \{d_1, d_2, \dots, d_i, \dots, d_n\}$. For each item, $t \in (t_1, t_2, \dots, t_m)$ denotes a set of feature items, and $tf_{i,j}$ counts the number of times that word j appears in the document \vec{d}_i . Thus, the texts can be represented as the matrix, where each row represents a specific document and each dimension of a column vector corresponds to a separate term feature. There are several different ways to compute the values of $tf_{i,j}$, which are also known as term weights. One of the best-known schemes is TF-IDF weighting.

For this method, we use a new variable to describe the word frequencies via a simple transform as in Eq. (5):

$$tf'_{i,j} = \log(tf_{i,j} + 1.0) \quad (5)$$

the transform in Eq. (5) has advantages when the number of features is zero or one, and it eliminates the effects of some large number of features. Another factor that would impact the classification decisions is the function words without actual meanings. This is because the function words normally do not contain specific information, which just increases the noise in parameter estimation. A heuristic transform in the information retrieval community, known as *inverse document frequency*, is widely used to discount terms by their document frequency [6]. A common way to do this is shown in Eq. (6):

$$tf''_{i,j} = tf'_{i,j} \log \left(\frac{N}{n_j + 1.0} \right) \quad (6)$$

where N represents the number of documents belonging to the training data sets and n_j represents the number of documents that contains *word* $_j$.

In practice, documents have strong word inter-dependencies. After a word first appears in a document, it is more likely to be there again. Because the Multinomial Naive Bayesian model assumes that the feature items in the document are independent of each other, it is harmful to the parameter estimation for long documents. This paper addresses this problem by normalizing the number of feature words. In practical terms, this paper uses a common IR transform [6] that has not been used with Naive Bayesian to eliminate the influence of word inter-dependencies. The formalization is shown in Eq. (7):

$$tf''_{i,j} = \frac{tf_{i,j}}{\sqrt{\sum_{j=1}^m (tf''_{i,j})^2}} \tag{7}$$

where m represents the number of feature items in the i^{th} document. Through combination and deformation for the above equations Eqs. (5), (6) and (7), we can obtain the following expression as Eq. (8):

$$tf'''_{i,j} = \frac{\log(tf_{i,j} + 1.0) \times \log\left(\frac{N}{n_j + 1.0}\right)}{\sqrt{\sum_{j=1}^m \left[\log(tf_{i,j} + 1.0) \times \log\left(\frac{N}{n_j + 1.0}\right)\right]^2}} \tag{8}$$

Meanwhile, because the proposed model in this paper is based on the Multinomial Bayesian Model, according to Eq. (4), $P(\vec{V}_c)$ is the priori probability of class c , which equals to the total number of words in class c divided by the total number of words in the training data set, and $W_{c,j}$ denotes the transformed posterior probability of word j in class c , which can be calculated via Eq. (9):

$$W_{c,j} = \log V_{cj} = \log P(w_j|V_c) \tag{9}$$

This model uses $P(w_j|V_c)$ to represent the probability of word j in the class c and uses variable n to count the total number of words in class c . $P(w_j|V_c)$ can be obtained from Eq. (10):

$$P(w_j|V_c) = \frac{N(w_j, V_c) + \delta}{\sum_{j=1}^n N(w_j, V_c) + \delta|v|} \tag{10}$$

for this equation, $N(w_j, V_c)$ represent the TF-IDF weight of $word_j$ in class V_c as Eq. (7) specified, when $N(w_j, V_c) = 0$, it denotes that $word_j$ does not exist in class c . Because the accuracy of classification will be degraded in this case, this defect can be optimized by Laplace Calibration [13] with a simple implementation. In this manner, the value of feature items that do not exist in the class will be added to a coefficient to avoid the probability being equal to zero. In Eq. (10), as δ is the Laplace smoothing coefficient, we can set $\delta = 1$ to achieve this effect, and $|v|$ denotes the size of different words in the class c . However, the

formula (10) does not take the number of words in all category into account. For instance, compared to the documents in category c_1 , if a category c_r has more documents, or the document has more words, the probability of $P(w_j|V_c)$ will likely be higher than the value in c_1 .

To address these above defects and shortcomings, we should consider the overall impacts of the proportions of feature words in documents, local classes, and training data sets. As an improvement, this paper introduces a comprehensive coefficient as an eclectic coefficient, as in Eq. (11):

$$P(w_j|V_c) = \left(\frac{N(w_j, V_c) + \delta}{\sum_{j=1}^n N(w_j, V_c) + \delta|v|} \right)^\alpha \times \left(\sqrt[\frac{\sum_{c=1}^m \sum_{j=1}^n N(w_j, V_c) + \delta|v|}{\sum_{c=1}^m T(w_j, V_c) + \delta|T|} \times \frac{T(w_j, V_c) + \delta}{\sum_{c=1}^m T(w_j, V_c) + \delta|T|}]{\beta} \right) \quad (11)$$

where $\beta = 1 - \alpha$ and α takes the different values and have different impacts on $P(w_j|V_c)$. In this manner, the text classification has been changed into a problem of searching for the maximum probability value of Eq. (11). The bigger α is, the greater the local influence of feature items is. Therefore, our target is to find the optimal value of α to achieve the best classification performance through multiple training. $T(w_j, V_c)$ counts the documents that contain the feature w_j in the class c . As in Eq. (10), we also set $\delta = 1$ with $|T|$ as the average number of documents for each class in the training data set in Eq. (11).

3.3 Noise Elimination

Function words are ubiquitous and enormous in general documents, have no benefit on the document classification, and may even cause some serious influence on the classification results. In this section, we proposed a method to implement noise reduction based on *RDD* before data training. After the original input data set is uploaded to HDFS, the data path named as *datasetsPath* will be used as the input of the following preprocess function: *preprocess()*. If the *datasetsPath* is a directory, this algorithm will traverse it, and then continue recursively calling the function *preprocess()*. If the *datasetsPath* is a file, a parallel noise elimination process *removingFuncWords()* based on Spark will be executed to remove the meaningless function words in the original data set. In this function, firstly we read files from HDFS according to the specified data path to obtain a *HadoopRDD*, then, we spilt the very line of documents into different words by using the *map()* function to get a *MappedRDD*, secondly a *filter()* function will be applied to previous *FilteredRDD*, to remove all words that appear in function words dictionary, finally a final *ProessedRDD* without function words will be returned. The specific process is shown in the Fig. 1.

The quality of the text in the training data set directly determines the result of the text classification. In practical applications, the construction of the training set inevitably produces noise samples. Besides some functional words which has simple structure contained in these samples, other kinds of words such as stop words, low frequency words, modal words... will also have a serious influence on the accuracy of the classification, because these words almost exist in

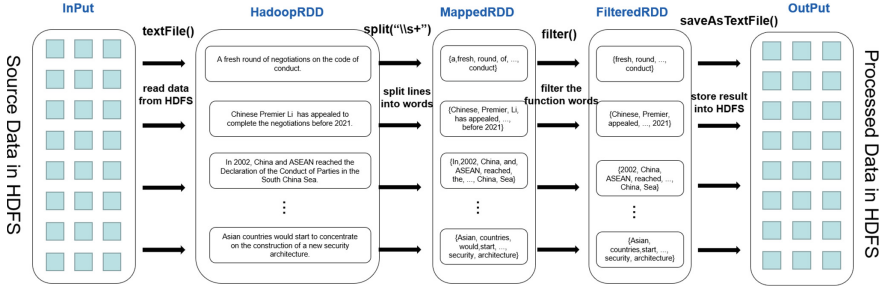


Fig. 1. The procedure of removing function words

all texts, and contain little information, and their function are too superficial to support the results of the classification. So we need to remove these words in each document of the training data set before text categorization, and then convert the remaining words into feature vectors. The processing of text noise is a very critical phase for text categorization, whose importance is self-evident. The DFRF (Document Frequency Ratio Filter) algorithm is a relatively simple algorithm with low computational complexity. It is mainly used for feature extraction and keyword filtering. The formula is as follows in Eq. (12):

$$DFRF(t, C_i) = \left\{ t \left| \log_2 \left(\frac{S_{(i,t)}}{\sum_{t=1}^n S_{(i,t)}} \times \frac{n_i}{N - n_i} \times \frac{C_{(i,t)}}{C'_{(i,t)}} \right) > \theta \right. \right\} \quad (12)$$

where $S_{(i,t)}$ represents the amount of words t in class i , n_i represents the total size of articles in class i , and N represents the size of articles in the whole training set, and $C_{(i,t)}$ denotes the number of articles which contains the term t in class i , $C'_{(i,t)}$ denotes the total number of articles containing the word t in other classes, θ denotes the threshold, and only words whose $DFRF$ rates exceeds a certain threshold can be chosen. We can use this method to filter out the noise data in the text during the preprocessing stage.

3.4 Dimension Reduction

Information Gain (IG) has been widely used in machine learning and acts as a function to measure the amount of information obtained from category prediction by judging whether a feature term is present in a document or not. Because Chinese characters are connected to each other in the text and only the combination of characters can express complete semantics, IG is not suitable for reducing the dimensions of Chinese documents. In this section, we take a method based on IG through the maximum entropy model and TextRank [14] to extract feature items to reduce the dimensions for Chinese documents.

This section illustrates the steps for English documents and Chinese documents. For English documents, we calculate the IG values of all terms in the preprocessed data set and extract the important terms with large Information Gain values. We will traverse the *ProcessedRDD* and calculate the IG value of each words according to Eq. (13), if the IG value of word is lower than given threshold, the word will filter out from the *ProcessedRDD*, so we can get the a *ResultRDD* after dimensions reduction.

The computational procedure of Information Gains is shown in Eq. (13):

$$\begin{aligned}
 IG(t) &= H(C) - H(C|t) \\
 &= - \sum_{i=1}^m P(C_i) \log_2 P(C_i) + P(t) \sum_{i=1}^m P(C_i|t) \log_2 P(C_i|t) + P(\bar{t}) \sum_{i=1}^m P(C_i|\bar{t}) \log_2 P(C_i|\bar{t})
 \end{aligned} \tag{13}$$

In Eq. (13), m is the number of classes in the preprocessed data set, $H(C)$ and $H(C|t)$ represent the original entropy of the system and the conditional entropy, respectively.

For the data sets of Chinese documents, this section implements the dimension reduction process through a retrieval algorithm based on the *Double Array Trie* data structure. *Double Array Trie* is essentially a deterministic finite automaton (DFA). Every node represents a state of automation, and the state transition among nodes depend on different Chinese character variables that are appended to the current *Double Array Trie*. When the end states in this DFA are reached, or the current state cannot transition to any other state, the retrieval is finished.

Based on *Double Array Trie*, we can use two arrays to store the actual data. One is named *base array*, in which every element represents the node of *Double Array Trie*. Another is named *check array* in which every element represents the previous state of the current state in this array. Equation (14) formalize the transfer from state s to state t :

$$\begin{aligned}
 base[s] + c &= t \\
 check[base[s] + c] &= s
 \end{aligned} \tag{14}$$

where variable c represents one or more input chars. The positions of Chinese characters in the *base array* should be depend on their Chinese encoding. If there are n words for which the first character is i , and the second characters of these n words are a_1, a_2, \dots, a_n , respectively, then the positions of the second characters in the *base array* should be $base[i] + a_1, base[i] + a_2, \dots, base[i] + a_n$. If the values of $base[i]$ and $check[i]$ are equal to 0 simultaneously, it denotes that the state does not exist.

After completing the words net based on *Double Array Trie*, we use the *Viterbi* algorithm to generate a word graph. Algorithm 1 describes this processing.

Algorithm 1. Viterbi Segment Algorithm

Require:

The words graph created by *DoubleArrayTrie* algorithm, which is named *WordNet*;

Ensure:

The list of shortest path of words graph;

Receive the Vertex list of words graph:*nodes*;

for $i = 1$ to $\text{lenthof}(\text{nodes})$ **do**

// The *nodeArray* is the list that contains vertex

nodeArray = $\text{nodes}[i]$;

for $j = 1$ to $\text{lenthof}(\text{nodeArrays})$ **do**

// The *node* is vertex

node = the j^{th} element of *nodeArrays*;

calculate the length of real word of *node*:*RWLength*;

for $k = 1$ to $\text{nodes}[i + \text{RWLength}]$ **do**

obtain *toVertex*: the j^{th} element of $\text{nodes}[i + \text{RWLength}]$;

calculate the shortest path from *toVertex* to *node*;

end for

end for

end for

create and return the list of shortest path depend on *nodes*.

3.5 Shuffle

The implementation of shuffle in Apache Spark consists of two kind of phase: *Shuffle Write* and *Shuffle Read*. In the *Shuffle Write* phase, the output of map task will be sorted by the partition id and key of record (key/value pair) in the memory buffer, when the capacity of buffer has arrived the threshold of memory, the intermediate data will be spilled into disk. In the *Shuffle Read* phase, the reducer tasks will fetch the corresponding intermediate data partly from different map nodes according to their own reducer id.

In the *Shuffle Write* phase, the input data is too massive to read into memory one time, it can only be loaded into memory partly, the result of calculation is stored in a data structure namely *Append only Map*, which is substantially a *data array* occupied with a continuous memory space. With the execution of map task, the capacity of *data array* will become bigger and bigger. Actually, there is a prediction process called *Maybe Spill Collection* before storing calculation result, which mainly predicts that whether this insertion will cause the out of memory (OOM) exception. Once this insertion is predicted to cause memory overflow, the *Spill Write* operation will be started up, and the data in *Append only Map* will be spilled into disk batch by batch, one batch may contain 10000 key/value pairs. After that, the *data array* in *Append only Map* will be cleared to leave more space for next round execution.

It can be seen from the above description that the prediction result of *Maybe Spill Collection* process determines the times of *Spill Write*, so how to predict memory overflow precisely is extremely important. However, the original prediction algorithm called *Moving average* algorithm is barely satisfactory.

This method only considers the latest and previous sampling of *Append only Map*, but the former samplings are all excluded which also have an significant effect on the estimated size of memory utilization of *Append only Map*. It also does not take the distribution of train data set into account, most data sets are not very uniform, especially for text data. So we apply a new algorithm called *Triple exponential smoothing* algorithm to predict memory capacity. Exponential smoothing is a rule of thumb technique for smoothing time series data using the exponential window function. It can make use of all the sampling results, the later the sampling is, the higher the weight will be. The every time of sampling sequence of *Append only Map* is represented by $\{x_t\}$ beginning at time $t=0$, and the output of the exponential smoothing algorithm is commonly written as $\{S_t\}$, which may be regarded as a best estimate of what the next memory size of *Append only Map* will be. The form of *Triple exponential smoothing* is given by the formulas Eq. (15):

$$\begin{cases} S_t^{(1)} = \alpha x_t + (1 - \alpha)S_{t-1}^{(1)} \\ S_t^{(2)} = \alpha S_t^{(1)} + (1 - \alpha)S_{t-1}^{(2)} \\ S_t^{(3)} = \alpha S_t^{(2)} + (1 - \alpha)S_{t-1}^{(3)} \end{cases} \quad (15)$$

where the $S_t^{(1)}$, $S_t^{(2)}$ and $S_t^{(3)}$ represent the single, double and triple smoothed value of memory size of *Append only Map* for time t respectively, α is the smoothing factor, and $0 < \alpha < 1$. we can calculate the feature memory size of *Append only Map* after T times sampling as in Eq. (16):

$$\begin{cases} x_{t+T} = A_t + B_t + C_t T^2 \\ A_t = 3S_t^{(1)} - 3S_t^{(2)} + S_t^{(3)} \\ B_t = \frac{\alpha}{2(1-\alpha)^2} \left[(6 - 5\alpha)S_t^{(1)} - 2(5 - 4\alpha)S_t^{(2)} + (4 - 3\alpha)S_t^{(3)} \right] \\ C_t = \frac{\alpha^2}{2(1-\alpha)^2} \left[S_t^{(1)} - 2S_t^{(2)} + S_t^{(3)} \right] \end{cases} \quad (16)$$

where x_t is the latest sampling size of *Append only Map*, when $T = 1$, the x_{t+1} is the predicted memory size of *Append only Map* of next update.

4 Parallel Implementation on Spark

Figure 2 illustrates the steps of training and testing in INBCS based on the Spark computing framework, which are presented in detail as follow:

Step 1. Read the preprocessed data sets from HDFS into *parallelCollection RDD* using the *textFile* function.

Step 2. Organize the intermediate data in the form of keys/values from *parallelCollectionRDD*, where the keys are specified as an identifier that consists of the class and document names and the values are the contents of documents.

$$\{\text{hello, morning, } \dots, \text{apple}\} \implies \{\text{class1/7834911 : hello, morning, } \dots, \text{apple}\}$$

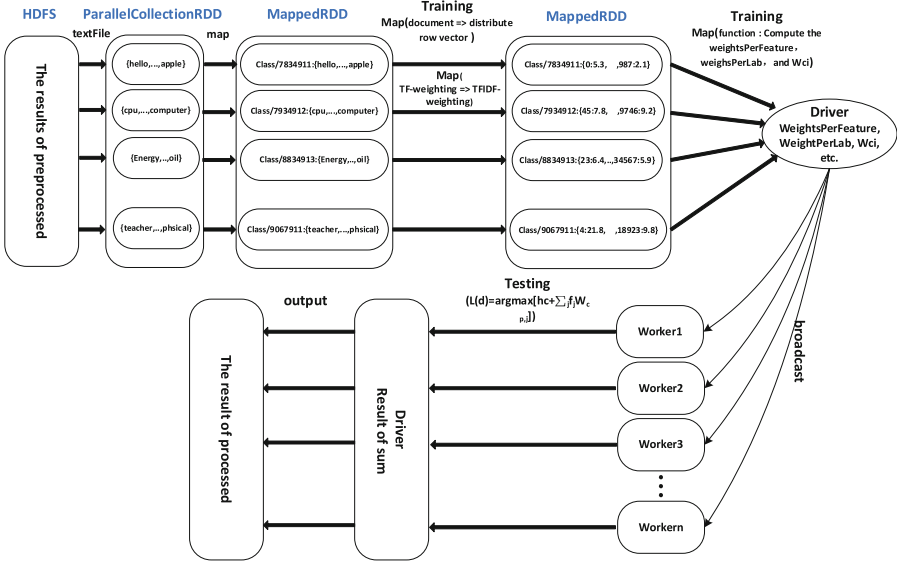


Fig. 2. The procedure of training and testing

Step 3. Transform the documents into distributed row vectors and convert the term frequency (TF) weight into the TF-IDF weight based on Eq. (8).

$$\{class1/7834911 : hello, morning, \dots, apple\} \implies \{class1/7834911 : 0 : 5.3, 1 : 7.8, \dots, 987 : 2.1\}$$

Step 4. Calculate some necessary parameters that can be used as the input to the text classification model. These parameters include *WeightsPerFeature*, *WeightsPerLabel* and $W_{c,i}$. They are described in detail as follows:

- (1) *WeightsPerFeature*: a one-dimensional vector in which the elements represent the weights of each feature word. Because the length denotes the number of feature words in data sets, it can be obtained by accumulating all the feature vectors for each document.
- (2) *WeightsPerLabel*: a one-dimensional vector in which each element represents the weight of all feature words in each class. The length of this vector equals the number of classes in data sets.
- (3) $W_{c,j}$: the posterior probability of word j in class c , which can be calculated by Eq. (9).

Step 5. Broadcast the global variables to all worker nodes.

Step 6. Test each worker node to determine which class their document should belong to according to Eq. (4).

Step 7. Collect the testing results and output the final results.

5 Evaluation

First, the posteriori probability of the term feature determines the classification accuracies. To estimate the impacts of various synthetic coefficients, we set a group of experiments to find the best coefficient to achieve the highest accuracy. Second, based on various sizes of data sets, this paper compares INBCS with a single-word-frequency algorithm (TF) and the TF-IDF weighting algorithm and then summarize the factors that affect the algorithm performances. Third, to evaluate the performances in various conditions, we run our experimental code on different numbers of computing nodes with various sizes of data sets in the Spark computation environment.

5.1 Experimental Settings

INBCS has been evaluated on a practical test cluster, which includes 6 slave nodes and 1 master node connected by a 1-Gb Ethernet switch. The experimental environment is based on Hadoop 2.6.0 and Spark 2.1.0. The hardware and software configurations are shown in Table 1. All experiments use the default configurations in Hadoop and Spark for HDFS.

Table 1. The software and hardware configurations in the Spark cluster

The node type	Master	Slave
Software environment	Ubuntu 12.04, JDK 1.7, Hadoop 2.6.0, Scala 2.11.0, Spark 2.1.0	Ubuntu 12.04, JDK 1.7, Hadoop 2.6.0, Scala 2.11.0, Spark 2.1.0
CPU	4 cores, 2.7 GHz	4 cores, 2.7 GHz
Memory	8 G	8 G
Quantity	1	6

For comparison, this section selects the following four widely used corpora in text categorization: the *Reuters21578* (*R8*), *20Newsgroups* data sets, *SogouLab_Reduced* data sets, and *answer* corpus. The *Reuters-21578* Distribution 1.0 dataset consists of 12,902 articles and 90 topic categories from the Reuters newswire. We use the standard ModApte train/test split. *20Newsgroups* (20NG) contains messages across twenty different UseNet discussion groups. *R8* and *20Newsgroups* are English data sets, while the *SogouLab_Reduced* corpus and *answer* corpus are Chinese data sets.

5.2 Performance Analysis

In our experiments, we select approximately 60% of the preprocessed data as training data sets by a random sampling method and take the rest as the testing

data sets. We can first acquire an optimal coefficient α from the Spark computing framework. This section compares the accuracies for the above four data sets with various values of α . In these experiments, we set $\alpha \in (0, 1)$, and the increment of the coordinate is 0.1. Figure 3 shows that the accuracies of these four data sets increase as the coefficient α varies from 0 to 0.9 and decreases as it varies between 0.9 and 1.0.

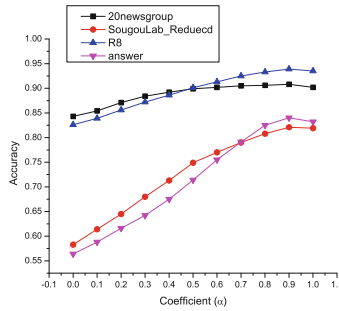


Fig. 3. The accuracies of different coefficient α

Because the time performances are usually related to the data quantities, the following experiments should estimate the execution time under the condition of different numbers of computing nodes with different sizes of input data sets. Using the same running environment from the above experiments, we choose the above four corpuses with size of 1 G, 5 G, 10 G, and 20 G.

As shown in Fig. 4(a) and (b), when the amount of data sets is 1 G or 5 G, the time consumption grows linearly as the number of computing nodes increases. This is because the Spark computing framework is based on memory computing. Each machine has 8 G memory; hence, a single computing node can load all the data sets into memory at once. Although the degree of concurrency of single node is lower than multi-node, the consumption time of the internal communication of the concurrent tasks is less than what the external nodes consume. Thus, this process can save a significant amount of communication consumption time among computing nodes. From these experimental results, when dealing with the small data sets, the performance of a cluster is usually lower than a single computing node with large memory.

As shown in Fig. 4(c) and (d), when the data size is 10 G or 20 G, with the increase of the number of machines, the processing time becomes less and less due to the limited memory capacity. In this situation, one machine cannot load all the data into memory. If there is only one node in the cluster, there will be many data exchanged between memory and disk. Meanwhile, the degree of concurrency is not high, which will lead to greater time consumption. Therefore, the time consumption by a single computing node is the largest. With an increasing number of nodes, the data sets can be distributed on more different nodes. After this, the increased number of nodes between clusters will yield increased

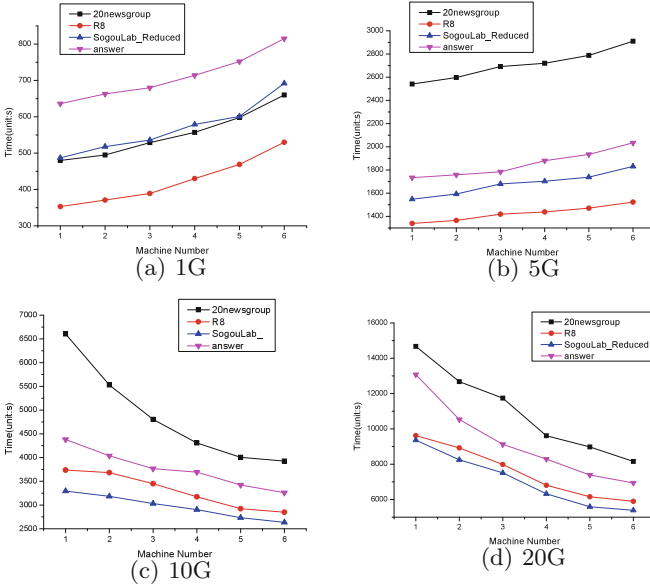


Fig. 4. Time consuming comparisons for INBCS in different dataset.

communication consumption. However, this is not the dominant factor of total time consumed. More nodes lead to a greater degree of concurrent tasks, and then the main factor in reducing the total time consumed is the higher degree of concurrency.

The experiments in Fig. 5(a) are all tested under a cluster with 8 nodes, and we also test the speedups of INBCS in different scales of the computing cluster. Figure 5(b) illustrates the speedups of INBCS with different corpus under the different node number of the cluster computing environment. From these results, we can easily draw a conclusion that the optimal speedup can be reached with 8 computing nodes in the experimental cluster. This is because the communication costs among the computing nodes usually increase as cluster scale increased. For the size of the input data in this experiment, eight computing nodes is an appropriate scale.

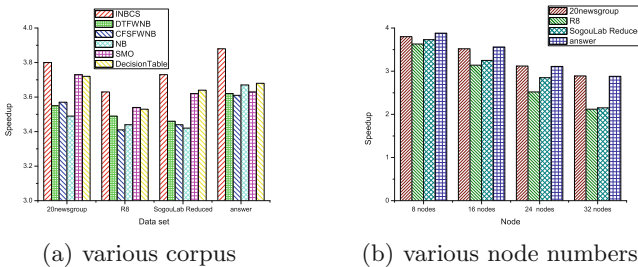


Fig. 5. The comparison experiments for the speed up.

In this section, a group of experiments is designed to compare the standard naive Bayes (NB) to naive Bayes with decision tree-based feature weighting (DTFWNB) [15] and naive Bayes with CFS-based feature weighting (CFS-FWNB) [16]. NB is regarded as the baseline.

All classification precisions were obtained by averaging the results from 8 separate runs of stratified 8-fold cross-validation. To further evaluate the performance of our proposed approaches over the standard baselines, the F1 values of classification are given in Table 2, which demonstrates that the competitive effects of INBCS are higher than NB, DTFWNB and CFSFWNB. The classification accuracies were obtained by averaging the results from 10 separate runs of stratified 10-fold cross-validation. From these results, INBCS can obtain satisfactory classification accuracies in these examinations.

Table 2. Classification F1 & Accuracy value comparisons

Dataset	F1				Accuracy			
	INBCS	DTFWNB	CFSFWNB	NB	INBCS	DTFWNB	CFSFWNB	NB
20news group	90.80%	86.47%	90.17%	86.79%	90.82%	87.69%	89.15%	86.79%
SogouLab Reduced	82.92%	82.83%	78.64%	78.34%	82.14%	81.15 %	76.15 %	78.58 %
R8	93.23%	93.46%	92.54%	90.68%	93.93 %	93.17 %	92.35 %	89.94 %
Answer	82.74%	82.64%	82.19%	81.48%	84.07 %	82.26%	82.47%	82.54%

To further validate the effectiveness of former proposed approach, INBCS is also compared with some other state-of-the-art methods that are not based on naive Bayes, such as SVMs using Sequential Minimal Optimization [17] and the Decision Tree [18]. In this experiment, INBCS is regarded as the baseline. Table 3 show the detailed comparison results in terms of classification accuracy, and AUC, respectively. To save time in running the experiments, we estimate the classification accuracy of each algorithm on each dataset by averaging the results from 5 separate tests of stratified 5-fold cross-validation.

Table 3. Classification Accuracy & UAC value comparisons

Dataset	Accuracy			AUC		
	INBCS	SVM	DecisionTree	INBCS	SVM	DecisionTree
20news group	90.82	90.27	87.49	97.45	93.35	92.94
SogouLab Reduced	82.14	83.26	69.45	96.17	95.79	88.89
R8	93.93	97.65	89.82	99.34	98.74	97.64
Answer	84.07	88.68	72.45	93.48	92.48	88.49

6 Conclusion

The work in this paper are all based on the improvement of Naive Bayesian model. Besides the noise elimination and high dimensionality reduction, this paper also proposes an improved method for calculating the conditional probability that comprehensively considers the effects of the feature items in each document, class and training set. Experiments on Spark clusters with large amounts of data confirmed that the method proposed in this paper can achieve better accuracy, F1 values and other performance evaluation indexes for several popular corpora. From these efforts we can draw the following insights: 1. For natural language processing, especially for text classification, a higher quality training data set is essential for classification, so, the pre-processing of the text on the early stage is very important. 2. Parallel computing may improve the computational efficiency certainly, however, it is very important to choose the appropriate cluster size according to the amount of data set, since the communication and data transmission between multiple nodes may consume heavy times, therefore, we need find an optimal point between the data size and the cluster scale. Although the algorithm proposed in this paper can improve the classification accuracy and computational efficiency to a certain extent, but the classification result of Chinese text is not very satisfied, it may be caused by the complicated structure of Chinese text and word segmentation, so we will improve the model deeply to adapt it to different scenes in the future.

Acknowledgments. The work is supported by the National Natural Science Foundation of China (Grant Nos. 61572176, 61873090, L1824034), the National Key Research and Development Program of China (2018YFB1701401, 2017YFB0202201), China Knowledge Centre for Engineering Sciences and Technology Project(CKCEST-2017-1-10, CKCEST-2018-1-13, CKCEST-2019-2-13).

References

1. Gudivada, V.N., Baeza-Yates, R., Raghavan, V.V.: Big data: promises and problems. *Computer* **48**(3), 20–23 (2015)
2. Apache Software Foundation. Spark (2015). <http://spark.apache.org>
3. Pernkopf, F., Wohlmayr, M., Tschitschek, S.: Maximum margin bayesian network classifiers. *IEEE Trans. Pattern Anal. Mach. Intell.* **34**(3), 521–532 (2012)
4. Bouboulis, P., Theodoridis, S., Mavroforakis, C., Evaggelatou-Dalla, L.: Complex support vector machines for regression and quaternary classification. *IEEE Trans. Neural Networks Learn. Syst.* **26**(6), 1260–1274 (2015)
5. Al-Mubaid, H., Umair, S.A.: A new text categorization technique using distributional clustering and learning logic. *IEEE Trans. Knowl. Data Eng.* **18**(9), 1156–1165 (2006)
6. Rennie, J.D.M.: Tackling the poor assumptions of naive Bayes text classifiers. In: *Proceedings of the Twentieth International Conference on Machine Learning*, pp. 616–623 (2003)
7. Aghdam, M.H., Ghasem-Aghaee, N., Basiri, M.E.: Text feature selection using ant colony optimization. *Expert Syst. Appl.* **36**(3), 6843–6853 (2009)

8. Shi, K., Jie, H.E., Liu, H.T., Zhang, N.T., Song, W.T.: Efficient text classification method based on improved term reduction and term weighting. *J. China Univ. Posts Telecommun.* **18**(18), 131–135 (2011)
9. Berka, T., Vajtersic, M.: Parallel rare term vector replacement: fast and effective dimensionality reduction for text. *J. Parallel Distrib. Comput.* **73**(3), 341–351 (2013)
10. Kim, S.B., Han, K.S., Rim, H.C., Myaeng, S.H.: Some effective techniques for naive bayes text classification. *IEEE Trans. Knowl. Data Eng.* **18**(11), 1457–1466 (2006)
11. Katz, S.M.: Distribution of content words and phrases in text and language modelling. *Nat. Lang. Eng.* **2**(1), 15–59 (2000)
12. Allison, B.: An improved hierarchical Bayesian model of language for document classification. In: *International Conference on Computational Linguistics*, pp. 25–32 (2008)
13. Meena, M.J., Chandran, K.R.: Naive Bayes text classification with positive features selected by statistical method, pp. 28–33 (2009)
14. Nie, Z., Zhang, Y., Wen, J.R., Ma, W.Y.: Object-level ranking: bringing order to web objects. In: *International Conference on World Wide Web*, pp. 567–574 (2005)
15. Hall, M., Hall and Mark: A decision tree-based attribute weighting filter for naive Bayes. In: Bramer, M., Coenen, F., Tuson, A. (eds.) *SGAI 2006*. Springer, London (2007). https://doi.org/10.1007/978-1-84628-663-6_5
16. Wang, S., Jiang, L., Li, C.: A CFS-based feature weighting approach to naive bayes text classifiers. In: Wermter, S., et al. (eds.) *ICANN 2014*. LNCS, vol. 8681, pp. 555–562. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11179-7_70
17. Platt, J.C.: *Fast Training of Support Vector Machines Using Sequential Minimal Optimization*. MIT Press, Cambridge (1999)
18. Kohavi, R.: The power of decision tables. In: Lavrac, N., Wrobel, S. (eds.) *ECML 1995*. LNCS, vol. 912, pp. 174–189. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59286-5_57