Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Performance evaluation of DNN with other machine learning techniques in a cluster using Apache Spark and MLlib

A.N.M. JayaLakshmi, K.V. Krishna Kishore *

Department of Computer Science and Engineering, Vignan's Foundation for Science, Technology and Research, Vadlamudi, Andhra Pradesh, India

A R T I C L E I N F O

Article history:
Received 7 March 2018
Revised 4 September 2018
Accepted 26 September 2018
Available online 28 September 2018

Keywords:
Sentiment analysis
Classification
Text mining
Big data
Distributed computing
Apache Spark framework

A B S T R A C T

Sentiment analysis on large data has become challenging due to the diversity, and nature of data. Advancements in the internet, along with large data availability have obviated the traditional limitations to distributed computing. The objective of this work is to carry out sentiment analysis on Apache Spark distributed Framework to speed up computations and enhance machine performance in diverse environments. The analysis, such as polarity identification, subjective analysis and email spam etc., are carried on various text datasets. After pre-processing, Term Frequency-Inverse Document Frequency (TF-IDF) and unsupervised Spark-Latent Dirichlet Allocation (LDA) clustering algorithms are used for feature extraction and selection to improve the accuracy. Deep Neural Networks (DNN), Support Vector Machines (SVM), Tree ensemble classifiers are used to evaluate the performance of the framework on single node and cluster environments. Finally, the proposed work aims at building an approach for enhancing machine performance, more in terms of runtime over accuracy.

1. Introduction

During the last decade, a wide interest has been generated in the area of sentiment analysis for applications that range from customer services to health care to marketing. Since the rapid growth of web and content generated on it being mostly in the format of text, text mining has become inevitable for various applications in different fields. This real time text data, mostly the unstructured data, which has been a challenge for processing and obtaining the user required results. In particular the challenges deals with people's experiences, opinions, and feelings. It is felt that the processing of text data, aided by machine learning techniques (Khan et al., 2016) can provide useful applications in various areas.

This explains the growing relevance of sentiment analysis in the last decade. For carrying out Sentiment analysis, data is crawled from various forms of web sources such as social media news articles, and live journals (Ahmed et al., 2015). Out of these, social media has attracted great attention when compared to other sources. Sentiment analysis involves use of natural language processing (NLP) techniques to systematically extract, identify subjective information, and study effective states from the crawled data. Hence, text analysis emphasizes on working with text data to know about things such as predicting the market value of a newly launched product based on surveys. Polarity detection is based on user reviews, movie ratings etc., Subjectivity & Objectivity analysis and email spam detections are based on the text contents in the mail. This dramatic growth in the area of sentiment analysis has made it possible to extend its applications to process huge amounts of Big Data. The speed with which volumes of the data is being generated from users through various sources is to be processed, which brings distributed programming into the frame. The application of the distributed programming technique helps in attaining scalability and resolves performance issues of various kinds of structured and unstructured data which cannot be handled by a single node system. Thus the blending of sentiment analysis with distributed frameworks can provide more effective results when compared to the existing models.

In this paper, we aim to compare the sentiment analysis with distributed environment (Burdorf, 2015) based on the Apache Spark Framework. Preprocessing and feature selection of the text from various datasets is done. Classification is performed using supervised machine learning algorithms, which includes working with ensemble techniques as well. We analyzed the performance

* Corresponding author at: Vignan's Foundation for Science, Technology and Research, VFSTR University, Vadlamudi, Guntur, Andhra Pradesh, India.

E-mail address: jayalakshmialuru94@gmail.com (K.V. Krishna Kishore).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2018.09.022
1319-1578/© 2018 The Author(s). Published by Elsevier B.V. on behalf of King Saud University.

of the model built in our work by focusing on the features extracted from input, rather than the accuracy.

The content in the article is arranged as follows. Section 2, presents a review of related works on the area of research. Section 3, discusses on the Distributed computing framework. Section 4 explains the Implementation of the work and the classification algorithms used in the proposed system. Finally, Section 5 presents an evaluation of the results obtained, while Section 6 draws the conclusions with suggestions for future scope in the area.

2. Related works

The tremendous increase of interest in the area of sentiment analysis is mainly due to the availability of data and the improvements brought in the era of the internet, advancements in new techniques and algorithms (Pang and Lee, 2006; Suttles and Ide, 2013; Bhavitha et al., 2017) has proved that blending sentiment analysis with machine learning can provide more scope of predicting success of newly launched products. The comparative study done on both supervised and unsupervised machine learning techniques reveal the efficacy of supervised techniques such as SVM over unsupervised techniques in sentiment analysis. Various methods and approaches for addressing the issues of sentiment analysis in real time, such as Polarity shift problem, binary classification, data sparsely, and accuracy are described in (Abirami and Gayathri, 2016).

Having such a wide range of applications of analytics, various forms of analytics like prediction, descriptive, prescriptive and diagnosis can provide immense advantage for real time big data analytics using Apache Strom, Hive, and HBase. Implementation of sentiment analysis using Apache Spark, has been done in (Baltas et al., 2017), where data crawled through Twitter has been classified using binary and ternary classifiers. Intensive machine learning algorithms on Spark have been evaluated (Svyatkovskiy et al., 2016). The problem of Policy diffusion detection problem (US) is considered as a use case, in which distributed text processing pipelines with spark data frames comprising Avro framework, Spark ML, Graph Frames, and Histogram suite are used.

Distributed analytics is now proving to play an important role in the real time applications (Oussous et al., 2017), which can indeed provide an improved facility of processing large scale data with minimal time complexity. The extreme learning framework, thus goes beyond big data analytics as proposed by (Oneto et al., 2016), in which Statistical Learning Theory (SLT) to build an Extreme Learning Machine (ELM). They used conventional ELM to work on big data using Spark which is deployed on Hadoop Cluster. A deep learning model (Alsheikh et al., 2016) is proposed in mobile big data analytics (MBD), and is a scalable learning framework using Apache Spark. The model proposes the parallelization of a deep model by slicing the MBD into many partitions in a spark resilient distributed dataset (RDD). The results indicate the attainment of a higher performance using deep learning models when compared to the conventional lighter models through the Spark framework on Actitracker dataset, which included accelerometer samples of 6 regular activities like jogging, walking, climbing stairs, standing, sitting, and laying down from the 563 crowd sourcing users.

An ensemble mail spam detection system using Apache Spark is implemented (Karthika Renuka et al., 2017). Target Prediction in discovering the Drug uses Apache Spark is proposed (Harnie et al., 2017), where remodeling of the existing pipeline has been implemented using Apache Spark for target prediction of drug molecule which damages proteins. SA's on social media such as Reddit, Live Journals, Twitter, etc., have been making tremendous applications in various fields. Nodarakis et al., 2016 proposed sentiment analysis using the Spark on large scale tweets. The hashtags and emoticons in tweets, are considered as input features and classified as sentiments in the distributed computing environment. The Bloom filter is used to compress the size and boost the performance as well. Various approaches proposed in sentiment analysis in literature are primarily focused on performance irrespective of time complexity. If data is small in size, analysis will be completed within time frame. But, when analysing large corpus, time is a major constraint, so the proposed framework will reduce the time taken for sentiment analysis.

3. Distributed computing environment

3.1. Map reduce model

Hadoop is a processing framework used to support the processing of large datasets in distributed computing (Bhosale and Gadekar, 2014). Map Reduce acts as a strong pillar in the Hadoop ecosystem. Map Reduce is a distributed and parallel programming model, which enables the processing of Big Data on a cluster computing environment. Map Reduce programming mainly consists of two functions, namely map () and reduce (). Mapper provides the mapping of input data according to the number of input partitions provided to the worker node, and generates a Key-Value pair as an output. Sort and Shuffle phase, provides the sorting of the data according to the given key input and generates the format readable for the Reducer.

The Reducer phase takes the input of these intermediate data and makes the transformations on the values for a given key value and generates the required output. The Apache Hadoop environment provides this feasibility of implementing the Map Reduce framework on top of it. Map Reduce has a disadvantage as it flushes the data to the disk between every iteration and the data is to be read on each iteration from the disk, which costs a lot in terms of time and disk I/O operations. This disadvantage has been overcome through the usage of Apache Spark which is a distributed programming framework developed to process the large volumes of data in the memory.

3.2. Apache Spark model

Apache Spark is a new framework which can provide an improved alternative to the Map Reduce model. Spark unlike map reduce model doesn't flushes out the data to the disk in each step, instead the data is processed in the memory until the memory becomes full. Once the memory is filled, then it spills over the data to the hard disk. Hence, Spark can also be referred as in-memory processing. This advantage of spark can make its processing very quick compared to map reduce models. Spark is generally referred to be 10x times faster than map reduce models. Spark framework can be implemented in various forms such as Standalone, on top of Hadoop Yarn (cluster manager), Cassandra or HBase. This forms another advantage of spark, which unlike map reduce doesn't require HDFS (Hadoop Distributed File System) to run. (Verma et al., 2016), instead spark can run on various other forms as mentioned above.

The architecture of the Spark is shown in the Fig. 1.The Spark Framework uses master/worker architecture where the worker nodes are managed by the master node. The executors in the worker nodes are implicitly built once the spark is deployed on the cluster, and the tasks are run as per the instructions given by the cluster manager (Eg: Yarn). The driver program is the user command interface through which the user sends the instructions and receives in the form of Spark Context or Spark Session. The Spark Session acts as the central gateway for spark communication from the spark versions 2.0 and above. At the core Spark works based on the concept of Resilient Distributed Dataset (RDD).

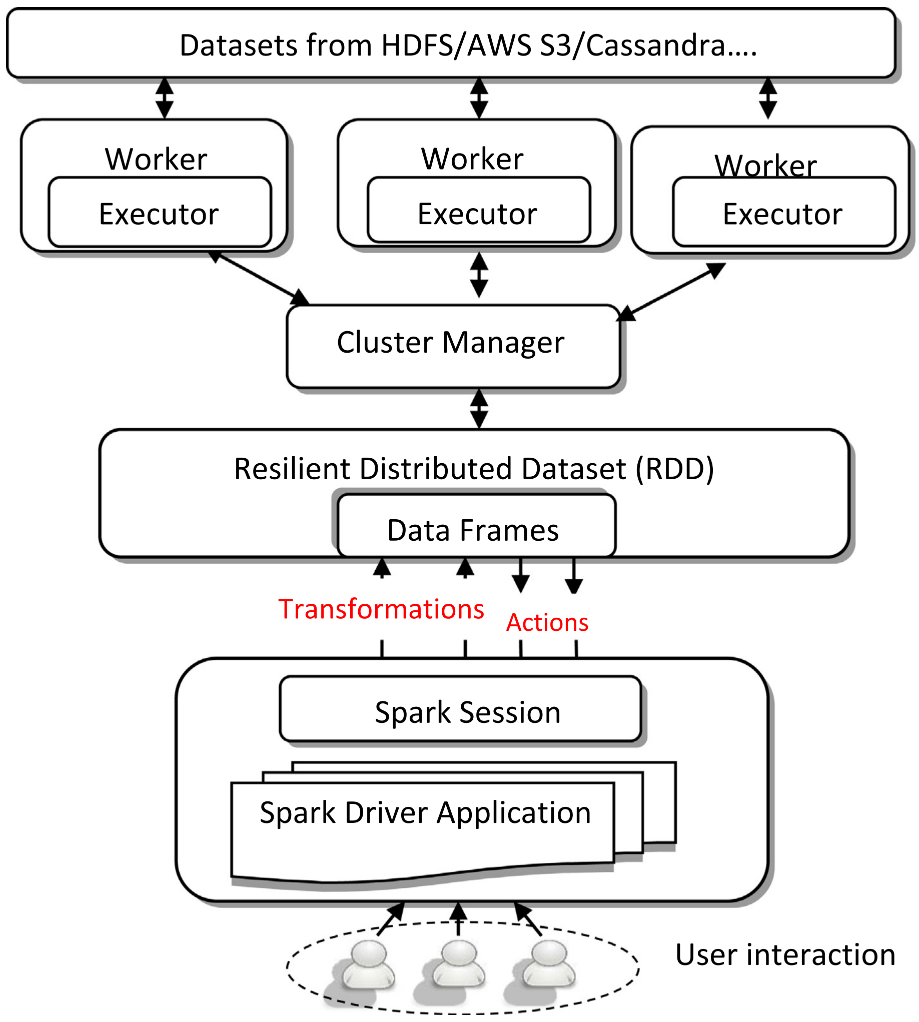

Fig. 1. Spark Architecture.

RDD's provide the implementations such as distributed data collection, fault-tolerance of the nodes, capability to use various data sources and parallelism. Our proposed model uses the data frames for storing and operating the datasets. Data frames provide a way more natural for working on tabular data. Although using data frames will have no impact on the implementation of the work, as the files will continue to be distributed as RDD's. It is only the syntax which changes. Spark performs two basic operations, i.e., transformations and actions. The transformations executes the work on the RDD's which converts the input data using operations such as map, join, reduce by key etc., and returns the output back to RDD's and actions collects the data from the RDD's.

3.3. Spark's MLlib

MLlib is the Sparks machine learning library which forms the implementation of machine learning techniques. It is a scalable machine learning library which can implement techniques of Classification, Regression, and Clustering. Spark is built on the Scala programming language, but API's such as Java, Python can also be operated on Hadoop or standalone environments.

4. Implementation

The architecture of the proposed model for implementation is given in Fig. 2. In the first phase data cleaning has been taken up, followed by extraction of the features for modelling. Then classification algorithms for classification of feature set are used in employing Spark MLlib to build the model. The first step of initial pre-processing is followed by the feature vector which is utilized by the classifier to classify the sentiment from each given text. Fig. 3 shows the overall workflow of the system. The distribution of the work across the cluster is shown. Considering the program execution as a job the spark divides the job into parallel partitions of RDDs. A spark job is referred as the computation of those partitions sliced into stages. The job in execution is split into N number of stages by the default Spark DAG Scheduler (Directed Acyclic Graph Scheduler). Each Stage has an id and the DAG Scheduler maintains the track of next Stage Id. A DAG scheduler thus splits a given job into a collection of stages where the required transformations and actions are consecutively performed. A Stage can have dependencies on other stages and therefore can trigger the execution of dependent stages. A Stage can only work on the partitions of RDD's. The partitions of each set of RDD's are divided into respective tasks (n number of tasks for n partitions) for each stage. When a Spark Context is created, each worker node starts an Executor. The Executor runs tasks on the nodes and returns back the results in the form of RDDs. The Result Stage is the final stage which tracks the completion of all the stages and returns the spark actions to the user program. The execution work flow of the proposed system on spark cluster also follows the same flow

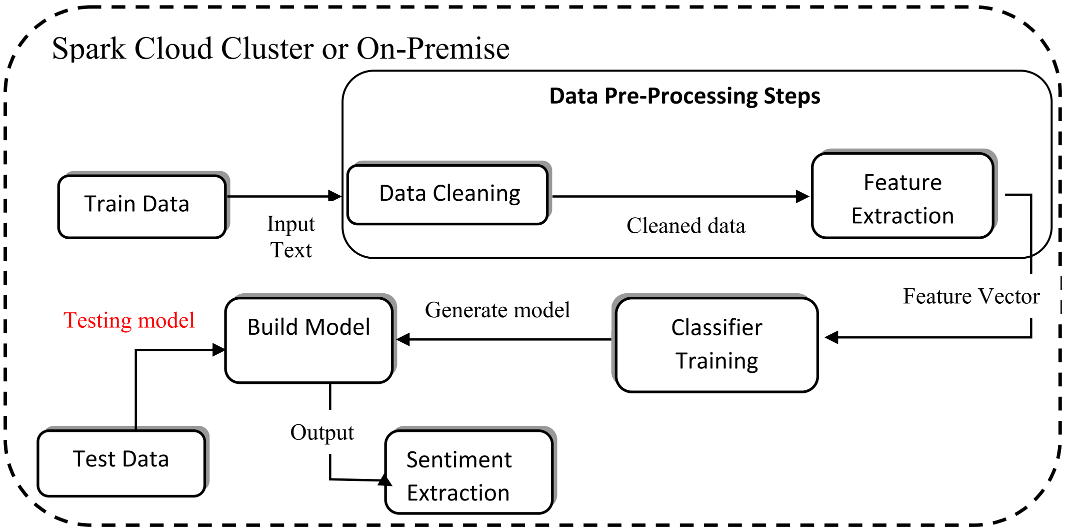

Fig. 2. System Overview.

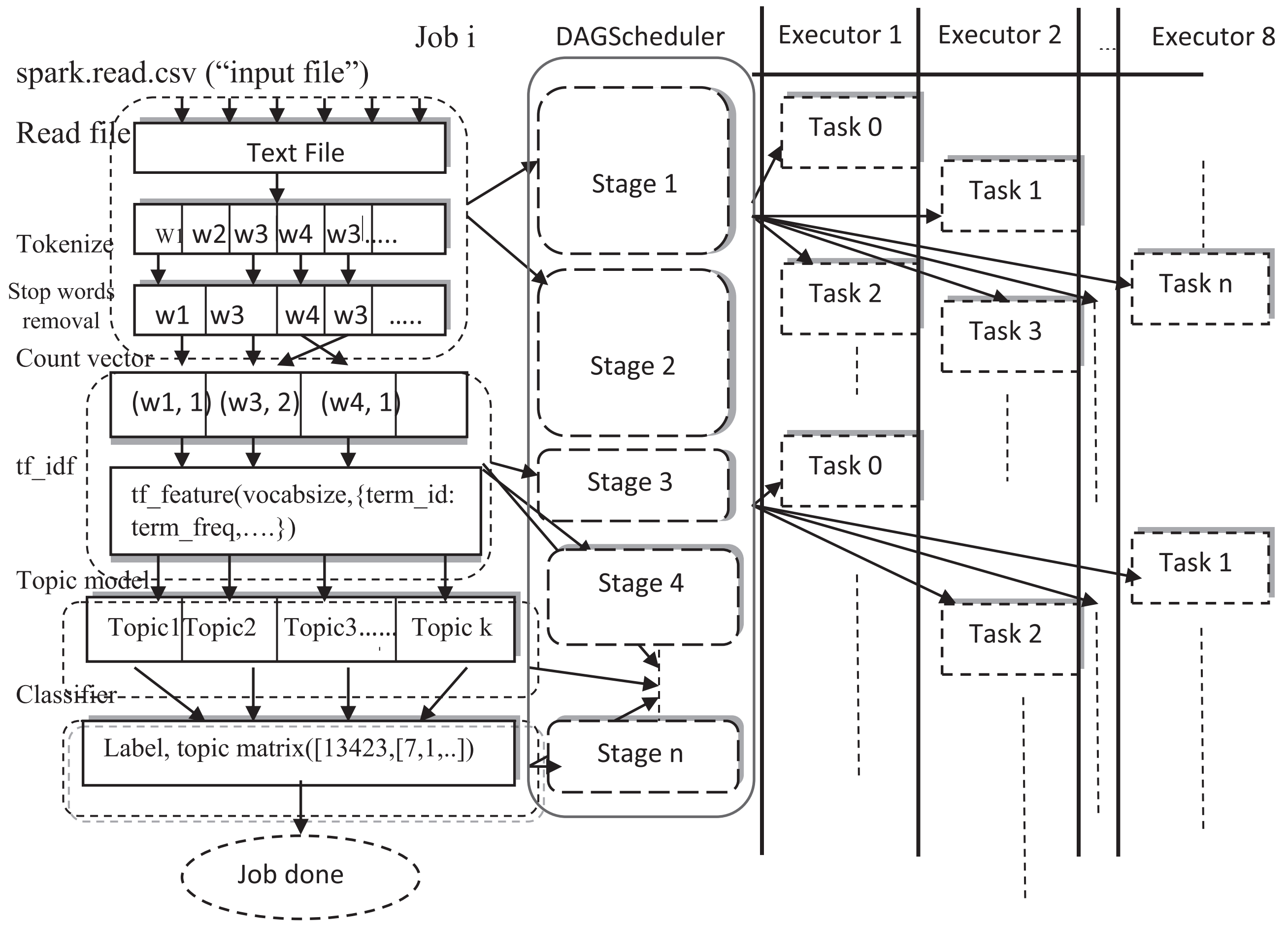

Fig. 3. Work Flow Overview.

as mentioned above. Considering the project as a sequence of 4 steps which requires the execution of one step followed by the other which involves reading of the data from the file followed by tokenization and stop word removal, Count Vectors and TF-IDF, topic modelling or LDA, classification. The respective steps are divided into N number of stages based on the user program by the default DAGScheduler which runs as N number of tasks on worker nodes by the executors. The stages are executed as tasks among the cluster of 8 worker nodes. The tasks are split randomly among the nodes and the transformations on the datasets are executed. Table 1 shows the methods used for spark operations such as transformations and actions on datasets.

4.1. Feature description

4.1.1. Extraction and Transformation

In this section, we show how the features are utilized to model the classifier. For the given text, we combine the feature to its respective feature vectors. We used a Data Frame-based API for the feature Extraction and Transformation which follows the implementation of Term frequency-Inverse document frequency (TF-IDF) (Liu and Yang, 2012). The TF-IDF is a vectorization method of features which is widely used in text mining. It reflects the importance of a term assigned to a document in the corpus. Term Frequency TF(t,d) where '*t*' is the term, '*d*' represents a document in a corpus 'D'. It refers to the number of occurrences of the term '*t*' in a document. Document Frequency DF(t,d) is the count of the documents which contains the term 't'. Whereas inverse document frequency is the numerical count of the amount of information, a term provides. TF_IDF is the product of TF and IDF.

$$\mathrm{IDF(t,d)} = \log((|\mathrm{D}|+1)/(\mathrm{DF(t,D)}+1)) \tag{1}$$

$$\mathrm{TFIDF(t,d,D)} = \mathrm{TF(t,d)} * \mathrm{IDF(t,D)} \tag{2}$$

where |D| is the total number of documents in the corpus.

Spark's MLlib uses TF and IDF models individually to obtain the features, where both Hashing Term Frequency and Count Vectorizer can be used to generate Term Frequency vectors.

4.1.2. Latent Dirichlet allocation in Spark

Latent Dirichlet allocation (LDA) is a topic model which refines topics from a collection of documents. LDA (Onan et al., 2016) is used for clustering of the documents, where the cluster centers refer to the topics from the given text data. This unsupervised machine learning algorithm is implemented through RDD-based Spark Clustering API which takes *Online LDA Optimizer* as the optimizer parameter. The parameter is an iterative mini-batch sampling and is much friendlier for distributed models. The output vector from the TFIDF is taken as the input of LDA topic modelling. From the extracted topics obtained through this unsupervised learning, we classify the data using supervised techniques.

4.2. Supervised Learning:

4.2.1. Naïve Bayes classifier

Naïve Bayes (Huang and Li, 2011) is a well-known multi class classifier. Spark MLlib has multinomial naïve Bayes and Bernoulli naïve *Bayes* classifiers. *Multinomial naive* Bayes (Jurafsky and Martin, 2016) is an approach for distinguishing the document classification. The term frequency tf(t,d) is calculated as

$$\text{Normalized term frequency} = \mathrm{tf(t,d)}/\mathrm{n_d} \tag{3}$$

Where, '$\mathrm{n_d}$' is referred as the number of terms in *d* document.

In the *multinomial* model, the class conditional probability P of the text x can be calculated as,

$$\mathrm{P}(\mathrm{x}|w_\mathrm{j}) = \prod_{i=1}^{m} \mathrm{P}(x_i|w_j) \tag{4}$$

The multivariate *Bernoulli naive Bayes* model is an alternate approach for classifying the text document on binary data. The feature vector has 'm' dimensions with the number of words referred as 'm' in the whole vocabulary (*The Bag of Words Model*). The value of 1 is the occurrence of the word in the particular document, and a value of 0 represents the absence of the word in the document. The Bernoulli trials are written as follows

$$P(\mathrm{x}|w_\mathrm{j}) = \prod_{i=1}^{m} P(x_i|w_j)^b \cdot (1-\mathrm{P}(\mathrm{x_i}|w_\mathrm{j}))^{(1-\mathrm{b})} \,\mathrm{where}(\mathrm{b} \in 0,1) \tag{5}$$

Table 1
Used Methods in the Model.

Method Name	Functionality
SparkSession.builder.appName('nlp')**.getOrCreate():**Spark Session is the entry point to work on Data frames in Spark. Creates a new Session or extracts an existing Session using the builder function.	Transformations
Spark.read.csv ("input text file", header = True, sep = '\t'): Reads a csv file data in the columnar format with header names. The *sep* parameter shows the column separator in the data file.	
Tokenizer (inputCol = "text", outputCol = "token_text"): Splits the word tokens from the given text file reading it line by line	
StopWordsRemover(inputCol = 'token_text',outputCol='stop_tokens', *caseSensitive = false*): Removes the commonly used words and also the words which provides no contextual meaning to the given text data as well. Case Sensitive Boolean attribute is set false by default to indicate that words are treated as case insensitive. The stop word list used is the default English stop words list which can be found from Glasgow Information Retrieval Group http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words.	
CountVectorizer():Counts the number of times a word occurs in the given document	
IDF (): Calculates the importance of the word in the whole document.	
StringIndexer():Assigns the numerical value to the string column.	
Pipeline (stages = [stage1, stage2, Stage n]): Pipelines the execution sequence of the instructions given.	
VectorAssembler(inputCols = ['tf_idf'],outputCol = 'features'):Assembler assembles the feature vector contents and provides the output features.	
LDA (featurescol="features",k = 10)**:** Provides topic modelling for the obtained feature vector based on the value of k given.	
df.select(['label'],['features']): selects the features and label columnar data from the available data frame	
ChiSqSelector(numTopFeatures = 10,featuresCol="features",outputCol = "selectedFeatures"): Chi-Squared feature selection decides top 10 features among the list of features.	
pyspark.ml.classification.DecisionTreeClassifier(): Classifies the data using decision trees algorithm. **pyspark.ml.classification. RandomForestClassifier(labelcol = 'label','featurescol = 'topicdistribution'):** Classifies the data using ensemble techniques using decision tree as base class. **pyspark.ml.classification. NaiveBayes():** Classifies the sample data to build the model **pyspark.ml.classification. MultilayerPerceptronClassifier():** To classify the data using back propagation neural networks.	
(training,testing) = Feature Vector.randomSplit([0.9,0.1]): The random Split method splits the pre-processed (described as Feature Vector in the System Overview) data in the ratio of 90 training samples and 10 testing samples out of given 100 data samples.	
Model = pyspark.ml.classification.MultilayerPerceptronClassifier().fit(training): Thefit method fits the training data to build the model using the provided classifier.	
Test results = model. Transform (testing): The transform method transforms the test data using the build model and forms the predicted labels.	
MulticlassClassificationEvaluator (metric Name = "accuracy").evaluate (test results): Evaluates the accuracy of the model predicted using the test results obtained by transforming the test data (described in the system overview) with the model build using the train data.	
Head (): It outputs the top 10 rows by default from the data frame.	**Actions**
Show (count, truncate = False): It outputs the transformations applied on the data to the user, with truncated values set to False in default.	
Collect(): It is used to collect the data from the data frame	

$$P^{\wedge}(x_i|\omega_j) = \left(df_{x_i,y} + 1\right)/(df_y + 2) \quad (6)$$

Empirical comparisons has proved that the multinomial model surpass the multivariate Bernoulli model if the size of the vocabulary is comparatively large since multivariate Bernoulli performs well with small vocabulary sizes (McCallum and Nigam, 1998).

4.2.2. Decision tree algorithm

The decision tree (Barros et al., 2012) follows a greedy method which performs a recursive binary partitioning of the features. The decision trees are considered as effective classifiers in classification, prediction, and for resolving decision-making problems. Decision trees are generally referred to as trees covering both classification and regression (or) CART. To maximize the information gain at a tree node argmax$_{s,}$ IG (D, s) is to be applied to a dataset D. The current implementation provides two impurity measures, which are Gini impurity and entropy. The implementation in the current work is made based on the current implementations done on spark 2.2.0 version. The current work uses the default parameters of spark framework which are *maxDepth* with a value of 5, *maxBins* (number of bins used for splits at each node) with a value of 32, impurity (default:gini)

$$\text{Gini impurity}: \sum_{i=1}^{c}(f_i(1-f_i)), \quad (7)$$

$$\text{Entropy}: \sum_{i=1}^{c}(-(f_i \log(f_i))), \quad (8)$$

where

- f_i is the frequency, C is count of unique labels.

4.2.3. Ensembles

Spark MLlib library supports notable ensemble algorithms, Gradient Boosted Trees (GBT) and Random Forest classifiers (RF) (Gupte et al., 2014). These GBT and RF ensemble models use decision trees as their base model. Random Forest Classifiers are found to be one of the best classifiers to solve real world classification problems (Fernández-Delgado et al., 2014). Random forest trains a set of decision trees individually, to implement the training in parallel. Combining the predictions from each tree reduces the variance which improves the performance of the test data. GBTs iteratively train decision trees to minimize the loss function and compare the predicted value with the labeled data. Random Forest algorithm works by randomly selecting K features from a total of M features where K<<M. The node d is selected among k features using best split algorithm. The daughter nodes from dode are further divided using the best split approach. The default K value of spark version 2.2.0 is used in the current work. The default parameters of the RFC, GBT in the spark implementations are used for classifying the proposed work. The default parameters set are *numTrees* (number of trees) default: 20, *maxDepth* (maximum depth of the tree) default:5, *maxBins* (number of bins used for splits at each node) default:32, *impurity* deafult:gini

Log Loss:

$$\sum_{i=1}^{N}(\log(1+\exp(-2\boldsymbol{y_i}F(x_i)))) \quad (9)$$

where

- N is number of instances,
- $\boldsymbol{y}$i = label of ith instance
- x_i is the features of ith instance,
- F(x_i) = predicted label of ith instance

4.2.4. Linear Support vector Machine

Support Vector Machines (SVM) (Mullen and Collier, 2004) is used for classification, regression and other tasks. SVM generates hyper planes for a multi-dimensional area. The hyper planes acts as the functional margin which reduces error rate for the classifiers. Spark's ML classifier supports binary classification with SVM using linear kernel function. The Linear SVC classifier of spark has outperformed well for the distributed frameworks which contains large size datasets. The default parameters set are *maxIter* (maximum number of iterations):100, *tol*(error rate) default:*1e-06.*

4.2.5. Deep learning

Multilayer Perceptron Classifier (MLPC) (Ain et al., 2017) is a classification based on the feed forward artificial neural network. MLPC contains multiple layers of nodes resulting in deep learning. Each layer is interconnected to the next layer forming a network. Feature vector is connected to the nodes of the input layer. Nodes of the hidden layer use sigmoid (logistic) function as the activation function or transition function (Mhaskar et al., 1994), whereas nodes of the output layer use the Softmax activation function. Deep Neural Network (DNN) implemented in the current work contains a total of 4 layers out of which the middle two layers are hidden layers and the first layer is the input layer comprising of the selected *numTopFeatures* (number of top features selected) number of neurons. The last layer is the output layer which contains two neurons based on the unique class labels in the given dataset. The number of neurons in the hidden layers are picked in random for each dataset to improve the accuracy. The default parameters of the Multilayer Perceptron Classifier () method of spark are used for the implementation of MLPC in the current work. The default parameters are *maxIter* (maximum number of iterations):100, *blockSize* (block size for stacking input data) default: 128, seed: 1234, *solver (algorithm) default:l-bfgs.*

$$\text{Sigmoid Function}: f(z_i) = 1/(1+e^{-z_i}) \quad (10)$$

$$\text{Softmax Function}: f(z_i) = e^{z_i}\left(\sum_{k=1}^{N} e^{z_k}\right) \quad (11)$$

where, N is the number of nodes in the output layers, which forms the number of the classes, and Z_i is calculated as $z_i = w_i x + b_i$ where w_i is the weight of ith node and b is the bias for each node.

5. Experimental evaluation

A series of experiments were conducted to evaluate the performance of the proposed model mainly under the perspective of running time and classification performance. Varieties of the datasets and a number of nodes are considered in analyzing the performance of the machine learning models. A Cluster is set up for experimentation with 8 computing nodes based on the size requirement of the datasets considered. Each one of them has four 2.4 GHz processors and 16 GB RAM, and 200 GB hard disks. On each node 64-bit Centos version 7 operating system has been installed. One of the nodes is considered as the master node and the other 7 nodes were considered as the slave nodes. Spark version 2.2.0 was set up on top of Hue. The Hue spark application is recently created which lets the users to interact directly with the spark application from any browser through any system. Pyspark library set up is installed with the required python API to run the applications on top of spark which is inbuilt with the Scala. Proposed model has been implemented on Spark Data frames which reads the data in the format of tabular values. The resilient distributed datasets (RDD's) then undergo the process of work distribution and fault tolerance to work on different nodes in the cluster.

Table 2
Precision, Recall, F-Score values of Email spam Dataset's.

Performance Metrics	Naive Bayes	Decision Trees	Random Forest	Gradient Boosted Trees	Linear SVC	DNN
Precision(single node Tf_idf)	94.98	95.01	95.96	96.98	97.96	95.40
Recall(single node Tf_idf)	91.57	92.34	93.32	93.02	97.91	95.49
F-Score(single node Tf_idf)	92.47	92.98	93.18	93.25	97.83	95.27
Precision(cluster Tf_idf)	94.46	94.45	95.6	95.88	90.96	95.4
Recall(cluster Tf_idf)	91.93	92.27	93.16	94.96	90.96	95.49
F-score(cluster Tf_idf)	92.57	93.22	94.11	94.11	90.96	95.27
Precision(single node LDA)	76.64	93.33	93.94	93.94	75.59	88.16
Recall(single node LDA)	87.81	93.95	94.48	94.83	86.78	88.78
F-Score(single node LDA)	81.81	93.27	93.80	94.10	80.53	85.61
Precision(cluster LDA)	72.55	92.99	93.17	92.97	85.9	88.15
Recall(cluster LDA)	85.37	93.12	93.12	93.12	87.72	88.60
F-score(cluster LDA)	78.25	92.81	93.01	93.78	86.88	85.46

Table 3
Precision, Recall, F-Score values of Subjectivity/Objectivity Dataset's.

Performance Metrics	Naive Bayes	Decision Trees	Random Forest	Gradient Boosted Trees	Linear SVC	DNN
Precision(single node Tf_idf)	89.25	88.2	88.89	89.25	89.27	88.05
Recall(single node Tf_idf)	89.24	88.51	89.12	89.58	89.26	88.04
F-Score(single nodeTf_idf)	89.24	88.51	89.23	89.11	89.27	88.04
Precision(cluster Tf_idf)	89.97	88.89	89.54	90.21	90.43	88.05
Recall(cluster Tf_idf)	89.96	88.13	90.43	90.96	90.4	88.04
F-score(cluster Tf_idf)	89.96	88.54	90.29	91.12	90.40	88.04
Precision(single node LDA)	65.50	65.38	68.15	68.75	64.63	67.90
Recall(single node LDA)	65.73	63.94	65.38	66.12	65.13	69.45
F-Score(single node LDA)	64.23	63.35	65.33	65.33	64.09	68.64
Precision(cluster LDA)	66.20	65.61	64.79	65.11	65.21	66.28
Recall(cluster LDA)	66.44	66.19	66.59	66.87	65.03	66.60
F-score(cluster LDA)	66.14	65.39	66.13	66.58	65.16	67.42

Table 4
Precision, Recall, F-Score values of Polarity Dataset's.

Performance Metrics	Naive Bayes	Decision Trees	Random Forest	Gradient Boosted Trees	Linear SVC	DNN
Precision(single node Tf_idf)	79.47	79.81	80.01	80.45	82.39	70.43
Recall(single node Tf_idf)	79.42	79.45	80.20	80.89	82.35	69.27
F-Score(single node Tf_idf)	79.41	79.94	80.16	80.54	82.35	69.12
Precision(cluster Tf_idf)	79.63	80.12	81.16	81.67	82.48	68.85
Recall(cluster Tf_idf)	79.55	80.45	80.98	81.63	82.46	69.80
F-score(cluster Tf_idf)	79.55	80.89	81.34	82.22	82.46	69.00
Precision(single node LDA)	51.67	53.76	54.84	54.14	53.23	51.99
Recall(single node LDA)	52.24	54.50	54.73	54.73	50.67	51.93
F-Score(single node LDA)	46.88	52.65	53.19	54.43	52.14	50.26
Precision(cluster LDA)	52.29	54.12	54.86	55.14	76.14	52.95
Recall(cluster LDA)	52.41	54.27	54.29	54.89	87.12	53.43
F-score(cluster LDA)	47.20	53.53	52.87	54.32	81.3	51.49

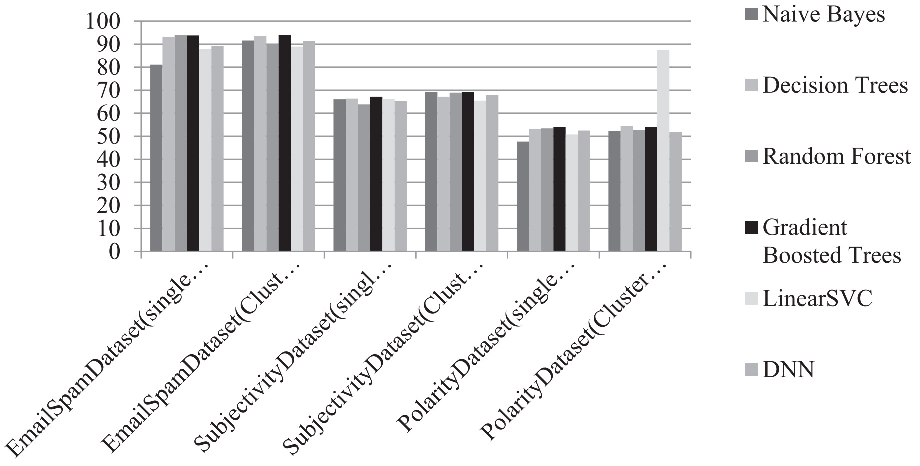

Fig. 4. Accuracy of Overall Dataset's with LDA.

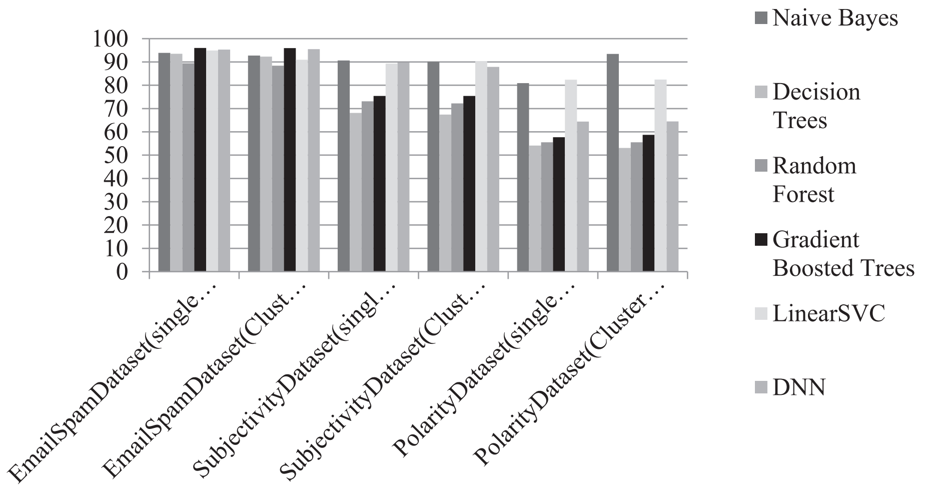

Fig. 5. Accuracy of Overall Dataset's with Tf_idf.

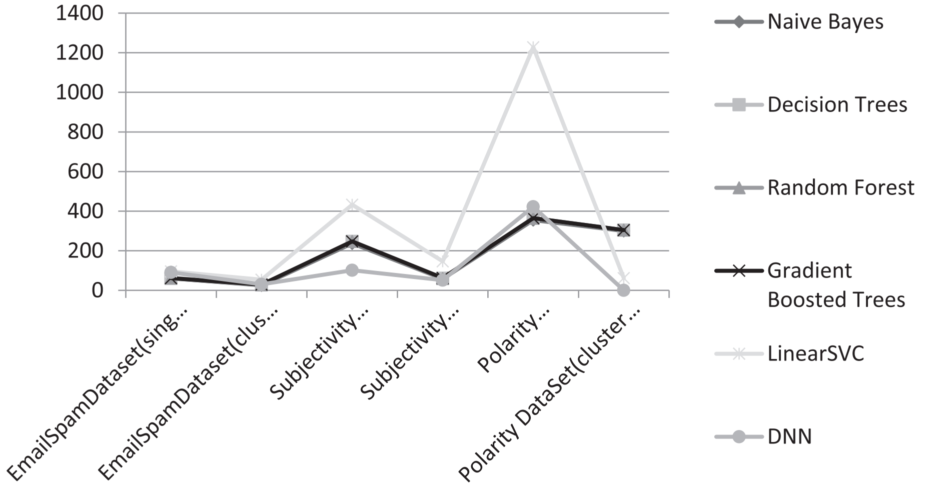

Fig. 6. Time Complexities of Overall Dataset's with LDA.

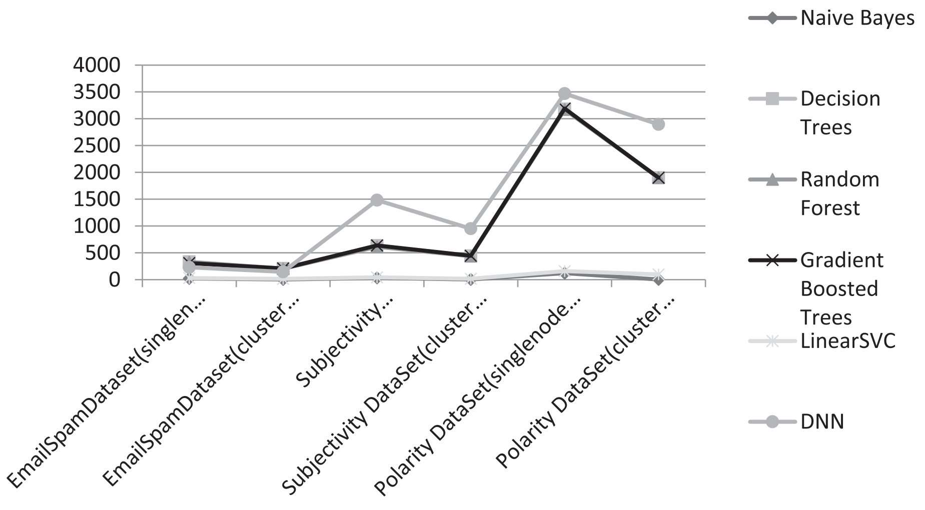

Fig. 7. Time Complexities of Overall Dataset's with Tf_idf.

5.1. Classification performance and running time

The classification performances of the models are evaluated in terms of accuracy using the random split method. The model trained using the ensemble classifiers outperforms the rest of the classifiers. The outputs generated through the ensemble trees techniques such as Random Forest and Gradient Boosted Trees show higher rates of accuracy. Assuming the fact that accuracy can also depend upon various influencing factors, classifiers with 55–60% of accuracy range can also be considered. In this framework, we have implemented the models which may not excel in terms of accuracy, but it is an effective model in terms of runtime.

Tables 2–4 shows the precision, recall and F-score performances of the model on the provided datasets. The values are considered on both cluster and single node systems. Figs. 4-7 shows the graphical representation of the performances on various classifiers. These graphs infers various performance metrics such as accuracy, runtime on various classifiers like Naïve Bayes, Decision Trees, Random Forest, Gradient Boosted Trees Ensemble techniques and Deep Neural Networks, Linear SVC. Fig. 4 shows the plotting of accuracy on overall datasets over various classifiers with LDA. The classifier accuracy on the single node versus cluster is showcased. Similarly the graphs in Fig. 5 show the performance metrics with TF-IDF on overall dataset's which shows the performance of the datasets over the single node and the Spark cluster. Figs. 6 and 7 shows the time complexity of models with LDA and with TF-IDF feature selection on various classifiers run on single node and cluster environments.

6. Conclusion and future work

In this proposed work, an efficient Spark framework with cluster is implemented to improve the time complexity in analyzing the sentiment analysis from the given corpus or from the real-time streaming. The text data from the corpus is taken as input in which features are extracted in the proposed model and further proceeded with clustering using LDA followed by the classification using various classifiers. The performance of the proposed framework using various classifiers is better on cluster when compared to the performance on a single node. The evaluation shows that the performances of DNN and LinearSVC, Tree Ensemble classifiers are better in the current work when compared to the rest of the classifiers. Through the evaluation, we have proved that our proposed system is scalable in terms of computation on large data and efficient. The limitation of the current work relies on LDA based feature extractions of sentiment analysis on distributed environments. The current availability of LDA on default spark versions is used in the work, which in turn has limited the accuracy of the model. The modelling of LDA with Rank based topic modelling can further improve the accuracy of the model.

In future, extraction of optimum features on high volumes of data will be considered. Furthermore, we will design a better machine learning model such as Generative Adversarial Network for improving performance under high volumes of data at a faster rate against other classification methods. The accuracy can be targeted in future for further implementation.

References

Khan, W., Daud, A., Nasir, J.A., Amjad, T., 2016. A survey on the state-of-the-art machine learning models in the context of NLP. Kuwait J. Sci. 43, 95–113.

Ahmed, K., Tazi, N., El Hossny, A.H., 2015. Sentiment analysis over social networks: an overview. 2015 IEEE Int. Conf. Syst. Man, Cybern., 2174–2179 https://doi.org/10.1109/SMC.2015.380.

Burdorf, C., 2015. A Distributed Sentiment Analysis. Development Environment.

Pang, B., Lee, L., 2006. Opinion Mining and Sentiment Analysis. Found. Trends® InformatioPang, B., Lee, L. (2006). Opin. Min. Sentim. Anal. Found. Trends® Inf. Retrieval, 1(2), 91–231. doi10.1561/1500000001n Retr.1,91–231 https://doi.org/10.1561/150000001.

Suttles, J., Ide, N., 2013. Distant supervision for emotion classification with discrete binary values. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 7817. LNCS, 121–136 10.1007/978-3-642-37256-8_11.

Bhavitha, B.K., Rodrigues, A.P., Chiplunkar, N.N., 2017. Comparative study of machine learning techniques in sentimental analysis ICICCT2017 Proc. Int. Conf. Inven. Commun. Comput. Technol., 216–221 https://doi.org/10.1109/ICICCT.2017.7975191.

Abirami, M.A.M., Gayathri, M.V., 2016. A survey on sentiment analysis methods and approach. 2016 Eighth Int. Conf. Adv. Comput. 72–76. https://doi.org/10.1109/ICoAC.2017.7951748.

Baltas, A., B, A.K., Tsakalidis, A.K., 2017. Algorithmic Aspects of Cloud Computing 10230, 15–25. https://doi.org/10.1007/978-3-319-57045-7.

Svyatkovskiy, A., Imai, K., Kroeger, M., Shiraito, Y., 2016. Large-scale text processing pipeline with Apache Spark. Proc. – 2016 IEEE Int. Conf. Big Data, Big Data 2016, 3928–3935. https://doi.org/10.1109/BigData. 2016.7841068.

Oussous, A., Benjelloun, F.Z., Ait Lahcen, A., Belfkih, S., 2017. Big data technologies: a survey. J. King Saud Univ. – Comput. Inf. Sci. https://doi.org/10.1016/j.jksuci.2017.06.001.

Oneto, L., Bisio, F., Cambria, E., Anguita, D., 2016. Statistical learning theory and ELM for big social data analysis. Eee Comput. Intell. Mag. 45–55. https://doi.org/10.1109/MCI.2016.2572540.

Alsheikh, M.A., Niyato, D., Lin, S., Tan, H.-P., Han, Z., 2016. Mobile Big Data Analytics Using Deep Learning and ApacheSpark, 22–29. https://doi.org/10.1109/MNET.2016.7474340.

Karthika Renuka, D., Visalakshi, P., Rajamohana, S.P., 2017. An ensembled classifier for email spam classification in hadoop environment. Appl. Math. Inf. Sci. 11, 1123–1128 https://doi.org/10.18576/amis/110419.

Harnie, D., Saey, M., Vapirev, A.E., Wegner, J.K., Gedich, A., Steijaert, M., Ceulemans, H., Wuyts, R., De Meuter, W., 2017. Scaling machine learning for target prediction in drug discovery using Apache Spark. Futur. Gener. Comput. Syst. 67, 409–417. https://doi.org/10.1016/j.future.2016.04.023.

Nodarakis, N., Tsakalidis, A., Sioutas, S., Tzimas, G., 2016. Large scale sentiment analysis on twitter with spark. CEUR Workshop Proc., 1558

Bhosale, H.S., Gadekar, D.P., 2014. A review paper on big data and hadoop. Int. J. Sci. Res. Publ. 4, 2250–3153.

Verma, A., Mansuri, A.H., Jain, N., 2016. Big data management processing with Hadoop MapReduce and spark technology: a comparison. 2016 Symp. Colossal Data Anal. Networking, CDAN2016. https://doi.org/10.1109/CDAN.2016.7570891.

Liu, M., Yang, J., 2012. An improvement of TFIDF weighting in text categorization. Int. Conf. Comput. Technol. Sci. 47, 44–47. https://doi.org/10.7763/IPCSIT.2012.V47.9.

Onan, A., Korukoglu, S., Bulut, H., 2016. LDA-based topic modelling in text sentiment classification: an empirical analysis. Int. J. Comput. Linguist. Appl. 7, 101–119.

Huang, Y., Li, L., 2011. Naive Bayes classification algorithm based on small sample set. 2011 IEEE Int. Conf. Cloud Comput. Intell. Syst 34–39. https://doi.org/10.1109/CCIS.2011.6045027.

Jurafsky, D., Martin, J.H., 2016. Naive Bayes and Sentiment Classification. Speech Lang, Process.

McCallum, A., Nigam, K., 1998. A comparison of event models for naive bayes text classification. AAAI/ICML-98 Work. Learn. Text Categ., 41–48 https://doi.org/10.1.1.46.1529.

Barros, R.C., Basgalupp, M.P., De Carvalho, A.C.P.L.F., Freitas, A.A., 2012. A survey of evolutionary algorithms for decision-tree induction. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 42, 291–312. https://doi.org/10.1109/TSMCC.2011.2157494.

Gupte, A., Joshi, S., Gadgul, P., Kadam, A., 2014. Comparative study of classification algorithms used in sentiment analysis. Int. J. Comput. Sci. Inf. Technol. 5, 6261–6264.

Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D., Amorim Fernández-Delgado, D., 2014. Do we need hundreds of classifiers to solve real world classification problems? J. Mach. Learn. Res. 15, 3133–3181. https://doi.org/10.1016/j.csda.2008.10.033.

Mullen, T., Collier, N., 2004. Sentiment analysis using support vector machines with diverse information sources. Conf. Empir. Methods Nat. Lang. Process. 412–418. https://doi.org/10.3115/1219044.1219069.

Ain, Q.T., Ali, M., Riaz, A., Noureen, A., Kamran, M., Hayat, B., Rehman, A., 2017. Sentiment analysis using deep learning techniques: a review. Int. J. Adv. Comput. Sci. Appl. 8, 424–433 https://doi.org/10.14569/IJACSA.2017.080657.

Mhaskar, H.N., Micchelli, C.A., 1994. How to choose an activation function. Adv. Neural Inf. Process. Syst. 6, 319–326.