Postprint: A Big Data Text Classification Method Using Distributed NBC in the Spark Framework

Authors: Zang Yanhui, Zhao Xuezhang, Xi Yunjiang

Date: 2018-10-11T00:00:00+00:00

Abstract

To address the challenges faced by existing big data-oriented computing frameworks in scalable machine learning research, this paper proposes a distributed Naive Bayes text classification method based on the MapReduce and Apache Spark frameworks. The proposed method explores the Naive Bayes Classifier (NBC) by investigating the adaptability of the MapReduce and Apache Spark frameworks, and studies existing big data-oriented computing frameworks. First, based on the Naive Bayes text classification model, the training sample dataset is divided into m classes. Furthermore, in the training phase, the output of the previous MapReduce is used as input to the subsequent MapReduce, employing four MapReduce jobs to derive the model. This design process fully leverages the parallel advantages of MapReduce. Finally, during classifier testing, the class label corresponding to the maximum value is selected. Experiments conducted on the Newgroups dataset achieved classification results exceeding 99% on all five categories of news data groups, outperforming comparison algorithms and demonstrating the accuracy of the proposed method.

Full Text

Preamble

Title: Text Classification of Big Data Using Distributed Naive Bayesian Classifier Under Spark Framework

Authors: Zang Yanhui[1], Zhao Xuezhang[1], Xi Yunjiang[2]
[1] Foshan Polytechnic, Foshan Guangdong 528137, China
[2] South China University of Technology, Guangzhou 510000, China

Abstract: Aiming at the challenges faced by existing big data-oriented computing frameworks in scalable machine learning research, this paper proposes a

distributed naive Bayesian text classification method based on the MapReduce and Apache Spark frameworks. The proposed method explores the naive Bayes classifier (NBC) by investigating the adaptability of MapReduce and Apache Spark frameworks, and examines existing big data computing frameworks. First, the training sample dataset is divided into m classes based on the naive Bayes text classification model. Further, during the training phase, the output of the previous MapReduce job serves as input to the subsequent MapReduce job, employing four MapReduce jobs to derive the model. This design fully exploits the parallel advantages of MapReduce. Finally, during classifier testing, the class label corresponding to the maximum value is selected. Experiments on the Newsgroups dataset achieved classification results exceeding 99% across all five news data categories, outperforming comparison algorithms and demonstrating the accuracy of the proposed method.

Keywords: text classification; MapReduce; Spark framework; distributed; naive Bayes classifier; machine learning

0 Introduction

Scalability has long been a critical research area in data mining and machine learning [1–4]. To tackle larger and more complex problems, algorithms continue to be refined for scale expansion. With ongoing hardware development and technological improvements, researchers face increasingly diverse challenges from emerging domains. In recent years, the rise of the internet economy and shifts in payment methods have created new development opportunities for traditional commerce, generating demand for storing massive amounts of data. This new data phenomenon, commonly termed "big data," opens a new window for machine learning [5–7]. Addressing these novel problems requires innovative solutions, leading to the emergence of various new computing frameworks suitable for highly distributed, data-oriented environments. Among existing proposals, MapReduce has gained tremendous recognition as a standard for processing massive datasets [8]. Since its inception, MapReduce has been actively developed for machine learning solutions, with specific techniques gaining popularity for framework adaptation. Reference [9] proposed an efficient and scalable kNN query model composed of multiple distributed phases. Apache Hadoop was established as the standard open-source implementation of MapReduce; however, Apache Spark has successfully introduced improved data abstractions, a faster execution environment based on main memory, and a user-friendly programming interface. Reference [10] presented a tree-augmented naive Bayes classification method based on the Hadoop framework, which partitions the training dataset into different blocks, distributes them across available compute nodes, and achieves high classification accuracy. However, this method suffers from high computational cost and difficulty in horizontal scaling.

This paper proposes a distributed naive Bayes text classification method based on the MapReduce and Apache Spark frameworks. The proposed method focuses on the learning phase of these models, for which we introduce a gen-

eral framework. This framework describes the complete family of naive Bayes classifiers (NBC) under the MapReduce paradigm and discusses the optimal characteristics and main limitations of these models from both theoretical and practical perspectives, aiming to identify future research directions with specially designed algorithms to address these issues. Experiments on the Newsgroups dataset achieved results exceeding 99% and outperformed comparison algorithms, demonstrating the accuracy of the proposed method.

1 MapReduce and Apache Spark Framework

1.1 MapReduce Framework

The MapReduce programming paradigm, designed by Google in 2003, is a big data processing tool for horizontal scaling [11–13]. Considered the most powerful search engine on the internet, it quickly became one of the most effective techniques for general-purpose data parallelization. Figure 1 [Figure 1: see original paper] illustrates the MapReduce data flow. MapReduce' s primary potential lies in its computational abstraction, where entire processing is divided into smaller task types—Map and Reduce—distributed and processed uniformly across the cluster. Practitioners need only provide these two functions, avoiding the need to adapt processing to the underlying cluster architecture or data nature. The framework provides a highly scalable, fault-tolerant environment for parallel data processing. The MapReduce process follows a two-step procedure where the cluster architecture is organized in a master/slave scheme: a master node configures jobs and distributes computational tasks and input data across a set of worker nodes that perform the processing. First, input data is split into smaller partitions or blocks, which are distributed among worker nodes as input for a given number of Map tasks.

The MapReduce paradigm is based on two separate user-defined primitives: Map and Reduce. The Map function reads raw data in the form of key-value pairs and transforms them into a set of intermediate key-value pairs, possibly of different types. Both key and value types must be user-defined. MapReduce then merges all values associated with the same intermediate key into lists (shuffle phase). Finally, the Reduce function takes the grouped output from mapping and aggregates it into a smaller set of pairs. This process can be schematized as shown in Figure 1 [Figure 1: see original paper]. This transparent and scalable platform automatically handles data in distributed clusters, relieving users of technical details such as data partitioning, fault tolerance, or job communication.

1.2 Apache Spark Framework

Apache Hadoop is the most popular open-source implementation of MapReduce for large-scale processing and storage on commercial clusters [14–16]. Due to its performance, open-source nature, installation facilities, and distributed file system (Hadoop Distributed File System, HDFS), this framework has become

ubiquitous across many domains. Despite Hadoop and MapReduce' s popularity, they demonstrate unsuitability in many cases, such as online or iterative computations. The inability to reuse data through memory primitives makes Hadoop infeasible for many machine learning algorithms.

Apache Spark represents a novel solution for large-scale data processing, considered capable of addressing Hadoop' s shortcomings. Introduced as part of the Hadoop ecosystem, Spark is designed to cooperate with Hadoop, particularly by utilizing its distributed file system. The framework proposes a set of in-memory primitives beyond standard MapReduce, aiming to process data more rapidly in distributed environments—up to 100× faster than Hadoop.

Spark is based on Resilient Distributed Datasets (RDD) [17], a special type of data structure for transparent parallel computation. These parallel structures enable persistence and reuse of results, caching them in memory. Additionally, partitioning can be managed to optimize data placement, and numerous transparent primitives can be used to manipulate data. All these features allow users to easily design new data processing pipelines.

The scalable machine learning library (MLlib) is built on Spark, thanks to its implicit suitability for iterative processes. The current version of MLlib (v1.6.0) contains numerous standard learning algorithms and statistical tools covering many important areas in the knowledge discovery process, such as classification, regression, clustering, optimization, or data preprocessing. MLlib is a key component of the MLbase platform, providing a high-level API that makes it easier for users to connect multiple machine learning algorithms. However, this platform does not include greedy learning algorithms such as kNN.

More specifically, beyond the horizontal strategy followed by most MapReduce-adaptive algorithms in machine learning, this paper also defines a fundamental parallel computational verticalization method. This vertical strategy can be further extended through intelligent procedures to balance data replication across different vertical Map tasks, thereby optimizing algorithm scalability for high-dimensional domains. According to the state-of-the-art Apache Spark computing framework, a specific implementation of this framework has been proposed. The software has undergone extensive benchmarking on diverse problem sets, with numerous performance metrics discussed. Beyond this benchmarking, several configurations of the computational architecture have been tested to provide a comprehensive overview of the scalability properties of our proposal. Figure 2 [Figure 2: see original paper] illustrates data transformations and operations in RDD during Spark execution.

2 Bayesian Network Classifiers

2.1 Notation

Given a vector of discrete predictive random variables $\mathbf{X} = \{X_1, \dots, X_n\}$ and a class variable C, we aim to derive a model from a training set D containing m

labeled examples, where C has domain $\Omega_C = \{c_1, \dots, c_l\}$. Table 1 summarizes the different symbols used in this study.

2.2 Bayesian Network Classifiers

Among other popular supervised classification models, Bayesian network-based classification models have gained increasing popularity in recent years. From a probabilistic perspective, given an example $\mathbf{x}$, the Bayesian classifier selects the value c^* that maximizes the posterior probability (MAP):

$$c^* = \arg\max_{c \in \Omega_C} P(c \mid \mathbf{x}) = \arg\max_{c \in \Omega_C} P(c) P(\mathbf{x} \mid c)$$

A Bayesian network (BN) is a probabilistic graphical model that enables efficient representation and manipulation of probability distributions. It is defined by two components: a graphical structure $G = (V, E)$ represented by a directed acyclic graph, where V is a set of nodes representing variables in A, and E is a set of edges encoding dependencies between nodes; and a set of numerical parameters Θ encoding quantitative information about variables and dependencies encoded in the graph. Specifically, for each variable X_i and its parent set $pa(X_i)$, a conditional probability distribution table (CPT) $P(X_i \mid pa(X_i))$ is stored. This representation encoded by BN can be used to recover the joint probability distribution due to the Markov rule:

$$P(X_1, \dots, X_n) = \prod_{i=1}^{n} P(X_i \mid pa(X_i))$$

Due to its solid mathematical foundation and predictive modeling capability, the Bayesian network formalism is widely used in many artificial intelligence tasks. However, general BN models have proven uncompetitive for supervised classification problems. For this reason, specific models have been proposed where the BN structure is adapted to handle this particular situation, typically treating the class variable as a node with a more important dependency role. Such models are commonly called Bayesian network classifiers (BNC) and are considered state-of-the-art methods in many domains.

Perhaps the most common BNC model is the popular naive Bayes (NB) classifier, where all predictive features in $\mathbf{X}$ are considered independent of the class C. Despite being a strong assumption that rarely holds in practice, the NB classifier' s performance has proven competitive for many problems, and its efficient training makes it a suitable choice for practitioners. This independence assumption defines a fixed structure for the model, eliminating the need to introduce correlations from training data. The NB classifier is trained on dataset D and its parameters are estimated using the maximum likelihood estimation (MLE)

algorithm and smoothing strategies such as Laplace smoothing, with computational complexity $O(mn\nu)$ and space complexity $O(ln\nu)$, which is also the space complexity of the resulting model.

2.2.1 Tree-Augmented Naive Bayes The tree-augmented model relaxes the conditional independence assumption by allowing each predictive attribute to depend on additional variables beyond the class. To select these specific dependencies, a structure learning algorithm is proposed that constructs a maximum spanning tree through conditional mutual information (CMI):

$$MI(X_i, X_j \mid C) = \sum_{c \in \Omega_C} \sum_{x_i \in \Omega_{X_i}} \sum_{x_j \in \Omega_{X_j}} P(x_i, x_j, c) \log \frac{P(x_i, x_j \mid c)}{P(x_i \mid c) P(x_j \mid c)}$$

Once the tree is obtained, a DAG is constructed by arbitrarily selecting a root node and orienting edges in topological order; finally, the class variable is added as a common parent to all nodes. This algorithm requires $O(mn^2\nu^2)$ computational complexity to calculate CMI for each attribute pair, for which a three-dimensional table with space complexity $O(ln^2\nu^2)$ is constructed. Building the maximum spanning tree has computational complexity $O(n^2 \log n)$. Parameter learning for the model requires $O(mn\nu)$ and space complexity $O(ln\nu^2)$, as it also builds three-dimensional tables. Computing the CPT for this specific network requires $O(mn\nu^2)$ and space complexity $O(ln\nu^2)$, where the occurrence counts for each possible configuration defined in the contingency table are stored in the given dataset D.

2.2.2 k-Dependency Estimator The k-dependency estimator can be viewed as a generalization of the previous idea, where each predictive variable in the model is allowed to have multiple k additional parents. This algorithm can explore a broader range of classifiers, starting from $k = 0$ (NB) and moving toward full BN as k increases. The classifier' s structure is learned through a three-step algorithm that also relies on mutual information. Beginning with the current predictive attributes, we construct a ranking σ between predictive attributes through their mutual information with the class variable C.

For each attribute X_i, we compute the conditional mutual information $MI(X_i, C \mid \mathbf{x}_\sigma)$. We then select the best k attributes with highest dependency as parents of X_i. Finally, the class variable C is added as a parent of all predictive attributes. Learning the network structure of a kDB classifier has computational complexity $O(mn^2\nu^2)$ and space complexity $O(ln^2\nu^2)$, as it also constructs three-dimensional tables. Computing the CPT for this specific network requires $O(mn\nu^{k+1})$ and space complexity $O(ln\nu^{k+1})$.

2.2.3 Averaged k-Dependency Estimator The Averaged k-Dependency Estimator (AkDE) adopts a slightly different approach. Instead of learning an augmented structure from the NB classifier, it avoids this costly operation by

defining a simple model ensemble with a fixed structure. For each model, all attributes depend on the class and an additional attribute called a super-parent. For a given k value, the AkDE classifier consists of n SPODEs, each with a different attribute X_i as super-parent. Classification is obtained by averaging the individual predictions from models in the ensemble.

Learning an AkDE classifier requires computing an $(k+2)$-dimensional table to estimate the required parameters, with computational complexity $O(mn\nu^{k+2})$ and space complexity $O(ln\nu^{k+2})$.

3 Distributed Naive Bayes Network Classifier

3.1 Distributed Naive Bayes Network Classifier

Based on the preceding analysis, the described algorithms share a common problem: the main computational burden of the learning process (structure or parameters) stems from computing multidimensional contingency tables from training data. If we use frequency-based methods such as MLE, the described metrics MI and CMI both require estimating probability distributions in the same manner as BN model parameter learning. Computing these contingency tables is the most demanding part of algorithms with complexity $O(mn^2\nu^2)$, where m corresponds to reading the complete training data and $n^2\nu^2$ corresponds to the dimensionality of the tables. For example, estimating parameters for an NB classifier requires learning two-dimensional tables involving each attribute (X_i) and class, while for an A1DE classifier, three-dimensional tables involving each attribute pair (X_i, X_j) and class are needed.

Generally, these contingency tables can be represented through histograms or frequency count distributions across different configurations of a given attribute set $\mathbf{X}$. We will compute the frequency $f_{\mathbf{X}}(\mathbf{x})$ of each occurrence of the joint state set of attributes in subset $\mathbf{X}$. The horizontal strategy divides data into blocks $D_1, \ldots, D_r$, ideally assigning them to r Map tasks. Each of these tasks computes partial contingency tables $f_{\mathbf{X}}^{D_j}(\mathbf{x})$ for each available local block D_j and each subset $\mathbf{X}$. Each Map task then emits a set of pairs $\langle \mathbf{X}, f_{\mathbf{X}}^{D_j}(\mathbf{x}) \rangle$ and associates them with their corresponding distributions.

In the Reduce phase, these pairs are grouped by their key (representing the attribute subset) and sent to the corresponding Reduce task, where the ideal number of tasks is $|\Psi|$, one per subset. The Reduce phase parallelizes aggregation of partial distributions for different attribute subsets and emits new key-value pairs containing the corresponding subset and its complete contingency table computed for the full dataset. By providing specific ranges of attribute subsets for Ψ, the previous framework can be instantiated for any described BNC.

The training sample dataset is divided into m classes, denoted as $\{C_1, C_2, \ldots, C_m\}$. The prior probability of class C_j occurring is denoted as $P(C_j)$, where $P(C_j) > 0$. The conditional probability that document d_i belongs to class C_j is $P(d_i \mid C_j)$. Then for any new document:

$$P(C_j \mid d_i) = \frac{P(C_j)P(d_i \mid C_j)}{\sum_{k=1}^{m} P(C_k)P(d_i \mid C_k)}$$

where $d_i = \{w_1, w_2, \ldots, w_v\}$ represents feature words, $i = 1, 2, \ldots, l$; thus k represents the total number of feature words; V represents the set of feature words.

The posterior probability of class C_j computed by Bayes' formula is expressed as:

$$P(C_j \mid d_i) \propto P(C_j) \prod_{k=1}^{V} P(w_k \mid C_j)$$

4 Experimental Evaluation

This paper addresses challenges in scalable machine learning research by proposing a distributed naive Bayes text classification method based on MapReduce and Apache Spark frameworks. The method focuses on supervised classification problems, exploring Bayesian network classifiers through the adaptability of MapReduce and Apache Spark frameworks. Experimental validation demonstrates the effectiveness of the proposed approach.

4.1 Running Environment

We utilize a computer cluster consisting of one master host and six slave nodes, each equipped with dual Intel Xeon E5-2609v3 1.90 GHz six-core processors and 64 GB RAM. Each worker node runs the HDFS file system on 4×1 TB disks and is managed by Cloudera CDH 5.5 distribution. In standalone deployment, the MapReduce environment is selected in Spark 1.6.0. We test algorithm behavior across different architectural layouts by launching different cluster configurations with varying resource quantities.

4.2 Classifier Training and Testing

We employ four MapReduce jobs to derive the model. The topology of multi-dimensional contingency tables must be configured, computed through correct identification of Ψ. In this case, the NB classifier encodes conditional probabilities for each attribute and class, defining two-dimensional tables with complexity represented by subsets $\Psi = \{\{X_i, C\}\} \forall X_i \in A$.

The output situations of three MapReduce jobs are shown in Tables 2 –4 . In the tables, *label* refers to the class label C_j, and *token* is the feature word w_k. The first MapReduce counts occurrences of $\langle label, token \rangle$ in the training set, computing term frequency (TF) values for each feature word per class. The second MapReduce calculates TF-IDF values for each feature word based on the output from Table 2. After computation completes, three files from the first MapReduce –featureCount, wordFrep, and termDocCount–are automatically deleted. The

fourth MapReduce computes results from the two $\langle label, token \rangle$ files in Table 3 and outputs the final results.

Based on previous calculations, the final mapper' s return value compares the test document's value under class C_j against values under other classes, selecting the class label corresponding to the maximum value.

4.3 Results Analysis

Experimental data adopts the 20 Newgroups dataset from the UCI KDD Archive [18–20]. During training, all documents from the Newgroups dataset classes are used. During testing, five news categories are randomly sampled: politics, baseball, religion, hardware, and motorcycle. These real-world datasets provide different attributes regarding scalability, summarized in Table 5 .

Table 5: Attributes of Real Datasets Included in Experiments

Category	Size (GB)
politics	
baseball	
religion	
hardware	
motorcycle	

Classification result comparisons are shown in Table 6 . The proposed method achieved classification accuracy exceeding 99% across all five news categories, outperforming comparison algorithms. The experiments demonstrate the accuracy of the distributed naive Bayes text classification method based on MapReduce and Apache Spark framework.

Table 6: Classification Result Comparison

Class Label	Test Docs	Correctly Classified	Literature [9]	Literature [10]
politics	234	234 (100%)	218 (93.16%)	229 (97.86%)
baseball	256	255 (99.61%)	246 (96.09%)	251 (98.05%)
religion	212	211 (99.53%)	201 (94.81%)	206 (97.17%)
hardware	166	166 (100%)	155 (92.81%)	156 (93.41%)
motorcycle	145	145 (100%)	141 (97.24%)	142 (97.93%)

5 Conclusion

This paper focuses on big data technologies such as MapReduce, proposing a solution suitable for current cloud computing and high-performance distributed programming paradigms. We analyze the most popular BNC models and their

corresponding learning algorithms, introducing a general computational framework that can be instantiated to learn any considered model. By targeting massive data with numerous examples or attributes, we increase elasticity and scalability. We extend our proposal with strategies for different computational cluster distributions. Experiments on the Newgroups dataset demonstrate superior results compared to alternative methods, proving the accuracy of our approach.

We also discuss the optimal characteristics and main limitations of these models from both theoretical and practical perspectives, aiming to identify future research directions with specially designed algorithms to address these issues.

References

[1] Gu Yuping, Cheng Longsheng. Classification of unbalanced data based on MTS-AdaBoost [J]. Application Research of Computers, 2018, 35(2): 346-348.

[2] Zhang Jifu, Li Yonghong, Qin Xiao, et al. Related-subspace-based local outlier detection algorithm using MapReduce [J]. Journal of Software, 2015, 26(5): 1079-1095.

[3] Huang Tinghui, Wang Yuliang, Wang Zhen, et al. Distributed traffic flow data prediction system based on Spark [J]. Application Research of Computers, 2018, 35(2): 405-409.

[4] Dang Hongen, Zhao Erping, Liu Wei, et al. Closed frequent item set mining based on data transformation and parallel computing [J]. Natural Science Journal of Xiangtan University, 2018, 40(1): 119-122.

[5] Baccarelli E, Cordeschi N, Mei A, et al. Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: review, challenges, and a case study [J]. Computers & Chemical Engineering, 2016, 91(2): 182-194.

[6] Zeydan E, Bastug E, Bennis M, et al. Big data caching for networking: moving from cloud to edge [J]. IEEE Communications Magazine, 2016, 54(9): 36-42.

[7] Sadeghi H, Valaee S, Shirani S. A weighted KNN epipolar geometry-based approach for vision-based indoor localization using smartphone cameras [C]// Proc of IEEE Sensor Array and Multichannel Signal Processing Workshop. [S. l.]: IEEE Press, 2014: 37-40.

[8] Hashem I A, Anuar N B, Gani A, et al. MapReduce: review and open challenges [J]. Scientometrics, 2016, 109(1): 389-422.

[9] Kalayeh M M, Idrees H, Shah M. NMF-KNN: image annotation using weighted multi-view non-negative matrix factorization [C]// Proc of IEEE Conference on Computer Vision and Pattern Recognition. [S. l.]: IEEE Computer Society, 2014: 184-191.

[10] Sun Kai, Kang H, Park H H. Tagging and classifying facial images in cloud environments based on KNN using MapReduce [J]. Optik-International Journal for Light and Electron Optics, 2015, 126(21): 3227-3233.

[11] Hyunseok C, Kodialam M, Kompella R R, et al. Scheduling in mapreduce-like systems for fast completion time [C]// Proc of INFOCOM. [S. l.]: IEEE Press, 2015: 3074-3082.

[12] Chen Rong, Chen Haibo, Zang Binyu. Tiled-MapReduce: optimizing resource usages of data-parallel applications on multicore with tiling [C]// Proc of International Conference on Parallel Architectures and Compilation Techniques. [S. l.]: IEEE Press, 2017: 523-534.

[13] Yigitbasi N, Willke T L, Liao G, et al. Towards machine learning-based auto-tuning of MapReduce [C]// Proc of IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems. [S. l.]: IEEE Press, 2014: 11-20.

[14] Murthy A C, Vavilapalli V K, Eadline D, et al. Apache Hadoop YARN: moving beyond MapReduce and batch processing with apache Hadoop 2 [Z].

[15] Wang Zhuo, Chen Qun, Li Zhanhuai, et al. An incremental partitioning strategy for data balance on MapReduce [J]. Chinese Journal of Computers, 2016, 39(1): 19-35.

[16] Xu Dezhi, Liu Yang, Sarfraz Ahmed. Optimization of RDF data storage and query based on Hadoop [J]. Application Research of Computers, 2017, 34(2): 477-480, 486.

[17] Yeager D S, Wang R. Comparing the accuracy of RDD telephone surveys and Internet surveys conducted with probability and non-probability samples [J]. Public Opinion Quarterly, 2015, 75(4): 709-747.

[18] Qi Fang, Feng Xi, Xu Qijiang. Direct push support vector machine classification algorithm based on artificial fish swarm optimization [J]. Computer Applications and Software, 2013, 30(3): 294-296.

[19] Chi Yunxian, Zhao Shuliang, Luo Yan, et al. Similarity measure for text classification based on feature subjection degree [J]. Computer Science, 2017, 44(11): 289-296.

[20] Trinh A T, Khambalia A, Ampt A, et al. Episiotomy rate in Vietnamese-born women in Australia: support for a change in obstetric practice in Viet Nam [J]. Bulletin of the World Health Organization, 2013, 91(5): 350-356.

Note: Figure translations are in progress. See original paper for figures.