Scalability Adaptation of Text Sentiment Analysis Algorithms in Big Data Environments: Using Twitter as a Data Source (Postprint)

Authors: Yu Chuanming, I understand the translation requirements. However, the provided text appears to be only "原赛", which seems incomplete. Please provide the full text to be translated, including all tags, LaTeX commands, and citations that need to be preserved., Wang Feng, An Lu

Date: 2023-07-26T00:00:00+00:00

Abstract

[Purpose/Significance] Aiming at the specific task of text sentiment analysis in big data environments, this study investigates the scale adaptation problem to provide guidance for information science researchers in achieving optimal trade-offs between efficiency and cost when conducting data analysis in big data environments. [Method/Process] Employing the Stanford Sentiment140 dataset and building upon the analysis of traditional sentiment analysis algorithms, we propose five text sentiment analysis algorithms tailored for big data, examine the adaptation performance of various algorithms across different environments and data scales, and conduct an empirical comparative study in terms of accuracy, scalability, and efficiency. [Results/Conclusion] Experimental results demonstrate that the cluster constructed in this paper exhibits favorable operational efficiency, correctness, and scalability; Spark clusters possess greater efficiency advantages in processing massive text sentiment analysis data, with this advantage becoming increasingly pronounced as data scale expands; regarding resource utilization, the overall operational efficiency of the cluster varies significantly with increases in node and core counts, and configuring five slave nodes with 4 cores and 4GB memory each enables efficient completion of classification tasks while achieving resource cost savings.

Full Text

Research on Scale Adaptation of Text Sentiment Analysis Algorithms in Big Data Environments: Using Twitter as a Data Source

Yu Chuanming[1], Yuan Sai[2], Wang Feng[1], An Lu[3]

[1]School of Information and Safety Engineering, Zhongnan University of Economics and Law, Wuhan 430073
[2]School of Statistics and Mathematics, Zhongnan University of Economics and Law, Wuhan 430073
[3]School of Information Management, Wuhan University, Wuhan 430072

Abstract

[**Purpose/Significance**] This study investigates the scale adaptation problem for the specific task of text sentiment analysis in big data environments, providing guidance for information science researchers to achieve optimal efficiency and cost trade-offs when conducting data analysis under big data conditions. [**Method/Process**] Using Stanford University's Sentiment140 dataset, we propose five text sentiment analysis algorithms for big data based on an analysis of traditional sentiment analysis methods. We examine the adaptation effectiveness of various algorithms across different environments and data scales, conducting empirical comparisons from the perspectives of accuracy, scalability, and efficiency. [**Results/Conclusions**] Experimental results demonstrate that our cluster architecture exhibits strong operational efficiency, correctness, and scalability. Spark clusters demonstrate greater efficiency advantages when processing massive text sentiment analysis data, with these advantages becoming more pronounced as data scale increases. Regarding resource utilization, cluster performance changes significantly with increases in node and core counts. Configuring five slave nodes with 4-core CPUs and 4GB memory achieves high classification efficiency while conserving resource costs.

1. Introduction

With the continuous development of mobile internet technology, the volume of data generated through online social and shopping activities has grown exponentially. Increasing numbers of users express opinions on entertainment, politics, and other events on social media platforms such as Weibo and Twitter. These opinions are typically disseminated through text, presenting new opportunities and challenges for both enterprises and governments in efficiently mining emotional information from massive text corpora [1]. In this context, the scale adaptation problem has emerged alongside domain adaptation [6-7] and language adaptation [8] as one of three major challenges facing opinion mining in big data environments [9].

It is worth noting that in addressing the scale adaptation problem for sentiment

analysis, two key issues require investigation: how the operational efficiency of sentiment analysis algorithms differs across various runtime environments, and how to select and configure optimal environments in practical applications. Currently, relevant research remains largely concentrated in computer science [10-13], while the scale adaptation problem for sentiment analysis has not received sufficient attention in information science and related fields. Given this gap, this paper analyzes traditional sentiment analysis algorithms and proposes text sentiment analysis algorithms designed for big data environments. We examine the adaptation effectiveness of these algorithms across different environments and data scales, conducting empirical comparisons of accuracy, scalability, and efficiency to provide guidance for researchers in related fields seeking optimal efficiency and cost trade-offs for text sentiment analysis in big data contexts.

2. Research Status

Research on sentiment analysis focuses on different aspects depending on data scale. For small-scale datasets, the emphasis lies on improving analysis granularity and extending applications to more scenarios. For large-scale datasets, researchers typically adopt statistical learning approaches, focusing on algorithmic improvements and parallel implementations.

On small-scale datasets, practical applications of text sentiment analysis include sentiment lexicon-based approaches, machine learning-based approaches, and extended applications. Sentiment lexicon methods extract sentiment words and their corresponding evaluation entities from text for polarity determination, including sentiment dictionary methods [14-15], ontology-based methods [16-17], and information extraction methods [18-19]. These studies typically involve fine granularity and substantial manual effort, making them suitable for smaller-scale data. Machine learning-based approaches bypass the cumbersome lexicon construction process by directly employing statistical learning methods [6,8,20-24], making them more adaptable to larger datasets. In terms of extended applications, sentiment analysis has been widely applied in intelligence fields such as customer review mining [25], online public opinion management [26-27], customer satisfaction surveys [28], online rumor identification [29], and competitor identification [30], where dependence on specific application scenarios outweighs considerations of data scale and algorithm selection.

On large-scale datasets, current research focuses on two main directions. First, algorithmic theory innovation combines big data platforms (e.g., Spark computing framework) to improve classification efficiency while maintaining or enhancing accuracy. For example, Zhu Jizhao et al. [31] designed a distributed CRFs algorithm—SparkCRF—for text analysis in Spark, demonstrating comparable accuracy to traditional single-node implementations while offering superior computational capability and scalability. J. Chen et al. [32] developed a parallel random forest (PRF) algorithm on Spark that employs dimensionality reduction during training and weighted voting during prediction, achieving better classification accuracy for large, high-dimensional, noisy data compared to Spark

MLlib algorithms. Second, horizontal comparison of classification algorithms on big data platforms seeks optimal algorithms for specific text data under different resource configurations, where classification performance is influenced by data attributes, dataset scale, and Spark resource allocation. Evaluation primarily compares classification accuracy and speedup ratios. For instance, G. Moghaddam et al. [13] compared decision trees, Naive Bayes, random forest, and SVM algorithms on airline datasets in single-machine Spark, finding decision trees outperformed the other three. A. Baltas et al. [1] used Spark MLlib for Twitter sentiment analysis, showing Naive Bayes achieved better classification results than logistic regression and decision trees for binary and ternary sentiment data, with dataset size significantly affecting Naive Bayes performance. M. Hai et al. [33] evaluated Naive Bayes and random forest performance in cluster Spark, identifying optimal node counts for different datasets and demonstrating high accuracy for both algorithms with varying speedup ratios across dataset sizes.

Against this background, this paper uses the Spark platform to investigate the computational efficiency, accuracy, and scalability of different sentiment analysis algorithms across various dataset scales and cluster configurations, aiming to provide recommendations for big data analysts on algorithm selection and platform configuration to achieve optimal efficiency and cost trade-offs.

3. Research Questions and Methods

3.1 Research Questions This study addresses the scale adaptation problem for text sentiment analysis algorithms in big data environments. By analyzing traditional sentiment analysis algorithms, we propose big data-oriented text sentiment analysis algorithms and examine their adaptation effectiveness across different environments and data scales from the perspectives of computational efficiency, accuracy, and scalability. Specifically, we pose the following research questions:

1. How does algorithm efficiency differ across runtime environments (traditional Sklearn, Spark single-node, and Spark cluster modes)? Does the Spark cluster mode offer greater advantages for massive sentiment classification data compared to Spark single-node and traditional Sklearn modes, with advantages becoming more pronounced at larger scales? How can runtime environments be better configured based on these differences when building big data sentiment analysis platforms?
2. How do various sentiment analysis algorithms perform across different runtime environments in terms of effectiveness (measured by accuracy, recall, F-value, etc.)? Does the Spark cluster mode demonstrate better scalability for massive sentiment classification data compared to Spark single-node and traditional Sklearn modes?
3. How does overall cluster efficiency change as node and core counts increase? How can optimal core and node counts be configured based on

these changes to achieve higher efficiency with lower overhead?

4. Below what dataset scale does the speedup ratio of parallel algorithms fall below 1, meaning cluster processing fails to demonstrate computational advantages? In other words, at what data scale does adopting parallel sentiment analysis algorithms become necessary?

This paper emphasizes text sentiment analysis for two reasons: First, research priorities differ across data scales—fine-grained analysis and scenario extension for small datasets versus algorithmic improvement and parallelization for large datasets. This study attempts algorithmic improvements and parallel implementations, focusing on sentiment analysis. Second, although emotion classification or polarity classification in text sentiment analysis can be framed as text classification problems, this paper uses Twitter as the data source, making sentiment analysis the primary task.

3.2 Big Data Text Sentiment Analysis Platform Architecture Current distributed computing platforms include Spark, Hadoop, GridGain, Mars, Phoenix, Twister, Disco, HaLoop, iMapReduce, iHadoop, PrIter, and Dryad [34]. Compared to other platforms, Spark's in-memory iterative processing offers better computational efficiency, fault tolerance, and scalability. Spark MLlib provides common machine learning algorithms and a Python programming environment, facilitating comparison between traditional Sklearn and parallel Spark implementations. Therefore, this paper adopts the Spark architecture for empirical research.

The big data text sentiment analysis framework is illustrated in Figure 1 [Figure 1: see original paper]. The system architecture comprises three components: Driver Program, Cluster Manager, and Worker Nodes [35]. The Driver Program is a client that defines SparkContext, requests resources from the Cluster Manager, and submits jar files and dependencies to Worker Nodes. The Cluster Manager allocates resources and tracks job status, serving as the Master Node in Standalone mode to control the entire cluster and monitor Worker Nodes. Worker Nodes (slave nodes) control compute nodes, including Tasks and Cache, launching Executors to perform computations and return results.

Experiments were conducted on the Qingyun server platform, with standalone and cluster environments configured as detailed in Table 1 . The standalone server used a 4-core CPU with 4GB memory, running traditional Sklearn and Spark single-node programs with Python 3.6. The cluster consisted of four node types: Client nodes (2-core CPU, 2GB memory) for storing local data and providing program interfaces; HDFS nodes (2-core CPU, 2GB memory) for user data storage and underlying data storage; Master nodes (4-core CPU, 4GB memory) controlling the cluster in Standalone mode; and Slave nodes as compute nodes with varying configurations (3-7 nodes, 4GB memory, 2/4/8 CPU cores) determining sentiment analysis algorithm efficiency.

3.3 Big Data Text Sentiment Analysis Algorithms Based on Decision Tree (DT), Logistic Regression (LR), Naive Bayes (NB), Random Forest (RF), and Support Vector Machine (SVM), we propose parallel versions of each algorithm. These algorithms were selected based on three principles: (1) Availability—widely used in traditional text sentiment analysis; (2) Scalability—capable of extension to large-scale data environments; (3) Comparability—offer comparable accuracy and efficiency. While other parallel classification algorithms exist (e.g., Gradient Boosting Trees, Multi-layer Perceptrons), they were excluded due to dependency issues between weak learners in gradient boosting and non-convex loss functions with multiple local minima in multi-layer perceptrons.

3.3.1 Parallel Decision Tree Algorithm The parallel decision tree employs a divide-and-conquer strategy, iteratively decomposing the original problem into multiple sub-problems with similar features. The algorithm includes three mechanisms: optimal attribute computation, tree node parallelism, and dataset parallelism [36]. Since node attributes are independent, information gain for each attribute can be computed by different processors, with the master processor determining the optimal attribute. Tree node parallelism divides tree nodes into independent sub-nodes that can be processed in parallel. The same decision tree algorithm processes data subsets across different processors, with classification models saved for subsequent use. Building on [36], our parallel decision tree algorithm is presented as Algorithm 1:

Algorithm 1: Spark-based Parallel Decision Tree Algorithm
Input: Dataset T
1. Read dataset from HDFS and convert to RDD
2. Use map operator to obtain sentiment scores and text vectors
3. Use Tokenizer operator to convert text vectors to word vectors
4. Compute TF-IDF for word vectors with feature count set to 3000
5. Obtain label and feature columns and index them to produce preprocessed dataset
6. Randomly split preprocessed dataset (80% training, 20% testing) with random seed 0
7. Fit decision tree classifier with max depth 10 to training data and record training time
8. Predict test data using trained model and output precision, accuracy, recall, and F1-score
Output: Sentiment analysis results

3.3.2 Parallel Logistic Regression Algorithm Parallel logistic regression is primarily based on Bagging sampling for parallelism, using limited-memory L-BFGS to accelerate coefficient estimation [12]. Bagging generates multiple classifiers with equal weights, aggregating them into a predictor through multiple voting. The predictor computes final predictions by integrating results from

multiple logistic regression classifiers. Building on [12], our parallel logistic regression algorithm is presented as Algorithm 2:

Algorithm 2: Spark-based Parallel Logistic Regression Algorithm
Input: Dataset T
1. Read dataset from HDFS and convert to RDD
2. Use map operator to obtain sentiment scores and text vectors
3. Compute TF-IDF values for text vectors
4. Use zip operator to connect TF-IDF vectors with sentiment scores and convert to LabeledPoint type
5. Randomly split converted dataset (80% training, 20% testing) with random seed 0
6. Fit logistic regression classifier with max iterations 1000 to training data and record training time
7. Predict test data using trained model and output precision, accuracy, recall, and F1-score
Output: Sentiment analysis results

3.3.3 Parallel Naive Bayes Algorithm Parallel Naive Bayes distributes training and test sets across multiple nodes for model construction and classification [37]. It computes TF and TF-IDF values for features in each class, calculates prior probabilities and TF-IDF sums per class, and generates conditional probabilities for features under each class. Building on [37], our parallel Naive Bayes algorithm is presented as Algorithm 3:

Algorithm 3: Spark-based Parallel Naive Bayes Algorithm
Input: Dataset T
1. Read dataset from HDFS and convert to RDD
2. Use map operator to obtain sentiment scores and text vectors
3. Use Tokenizer operator to convert text vectors to word vectors
4. Compute TF-IDF for word vectors with feature count set to 3000
5. Obtain label and feature columns and index them to produce preprocessed dataset
6. Randomly split preprocessed dataset (80% training, 20% testing) with random seed 0
7. Fit Naive Bayes classifier (smoothing=1, multinomial model) to training data and record training time
8. Predict test data using trained model and output precision, accuracy, recall, and F1-score
Output: Sentiment analysis results

3.3.4 Parallel Random Forest Algorithm Parallel random forest operates in two phases [38]. Phase one trains the classifier by converting vectorized training data to RDDs distributed across nodes, applying Bagging sampling (sampling count K determines tree count), building decision trees on samples, and aggregating distributed trees via union operations to generate the random

forest. Phase two classifies test data through voting across all decision trees, selecting the most frequent class as the final result. Building on [38], our parallel random forest algorithm is presented as Algorithm 4:

Algorithm 4: Spark-based Parallel Random Forest Algorithm
Input: Dataset T
1. Read dataset from HDFS and convert to RDD
2. Use map operator to obtain sentiment scores and text vectors
3. Use Tokenizer operator to convert text vectors to word vectors
4. Compute TF-IDF for word vectors with feature count set to 3000
5. Obtain label and feature columns and index them to produce preprocessed dataset
6. Randomly split preprocessed dataset (80% training, 20% testing) with random seed 0
7. Fit random forest classifier (10 trees, max depth 10) to training data and record training time
8. Predict test data using trained model and output precision, accuracy, recall, and F1-score
Output: Sentiment analysis results

3.3.5 Parallel Support Vector Machine Algorithm Parallel SVM training follows this process [39]: First, read HDFS data using RDD's textFile function and convert to RDD format, randomly splitting into appropriately sized blocks. Map operations perform parallel SVM training on formatted data, while repartition functions integrate results and partition data blocks for iterative SVM training until obtaining a globally optimal classifier. Building on [39], our parallel SVM algorithm is presented as Algorithm 5:

Algorithm 5: Spark-based Parallel SVM Algorithm
Input: Dataset T
1. Read dataset from HDFS and convert to RDD
2. Use map operator to obtain sentiment scores and text vectors
3. Compute TF-IDF values for text vectors
4. Use zip operator to connect TF-IDF vectors with sentiment scores and convert to LabeledPoint type
5. Randomly split converted dataset (80% training, 20% testing) with random seed 0
6. Fit SVM classifier with max iterations 1000 to training data and record training time
7. Predict test data using trained model and output precision, accuracy, recall, and F1-score
Output: Sentiment analysis results

3.4 Parameter Settings Parameter selection follows two principles: (1) Usability—parameters enable model usability with relatively good classification performance; (2) Equivalence—parameters remain as similar as possible across

environments (traditional Sklearn, single-node, multi-node) to enable effective efficiency comparisons. DT: 3000 features, max depth 10; LR: 1000 iterations; NB: 3000 features, multinomial model; RF: 3000 features, 10 trees, max depth 10; SVM: 1000 iterations.

3.5 Evaluation Metrics Evaluation comprises effectiveness and efficiency metrics. Effectiveness uses precision, accuracy, recall, and F1-score. Efficiency includes runtime and speedup ratio. Runtime measures training model generation time. Speedup ratio compares single-node versus parallel execution times for the same task, calculated by varying node counts while keeping datasets constant. T_1 represents single-node runtime, T represents runtime on m nodes. Linear speedup increase indicates effective time reduction through multi-node processing.

Speedup = T_1 / T

4. Experimental Results and Discussion

4.1 Experimental Data Massive text sentiment classification remains challenging in microblogging. Twitter's short-form content better expresses user emotional states. We selected Stanford University's Twitter sentiment dataset [40-41] based on two considerations: First, it has been widely used in text sentiment mining and social network analysis [42-45]. Second, it can be partitioned into different scales for scale adaptation research.

The dataset contains 1,578,627 tweets, with 1 indicating positive sentiment and 0 indicating negative. We extracted subsets of 15M, 30M, 75M, 150M, and 300M (corresponding to 160k, 310k, 780k, 1.58M, and 3.16M tweets) to study algorithm differences across scales. During preprocessing, 80% of data was used for training and 20% for testing, with each experiment repeated three times to average training time.

4.2 Experimental Results We evaluated five traditional algorithms (DT, LR, NB, RF, SVM) and their parallel counterparts across traditional Sklearn, single-node Spark, and cluster Spark environments.

4.2.1 Runtime Efficiency Experiment 1 compared runtime efficiency across environments using the 150M dataset. Traditional and single-node Spark used 4-core/4GB environments; cluster Spark used three 4-core/4GB slave nodes. Figure 2 [Figure 2: see original paper] shows runtime differences.

Runtime efficiency from low to high: Spark single-node, traditional Sklearn, Spark cluster. Spark single-node was slower than Sklearn due to PySpark's RDD conversion overhead versus Sklearn's direct numpy array operations. Spark cluster reduced runtime by nearly half compared to Sklearn, as Spark's RDDs store intermediate data in slave node memory, enabling direct memory reads across operations and significantly improving iterative performance.

4.2.2 Accuracy Experiment 2 compared accuracy across environments using the 150M dataset, evaluating precision, accuracy, recall, and F1-score. Figure 3 [Figure 3: see original paper] shows F1-score comparisons.

Except for DT and SVM, traditional Sklearn marginally outperformed Spark modes. For LR, Sklearn achieved F1=0.8019 versus 0.6181 (single-node) and 0.6172 (cluster), primarily due to different prediction data processing strategies. Spark single-node and cluster modes showed similar accuracy, indicating cluster expansion does not reduce classification accuracy. NB achieved the best performance with F1 values of 0.7153 and 0.7173, representing approximately 3% improvement over Sklearn's SVM.

4.2.3 Scalability Experiment 3 evaluated scalability by varying slave node core counts (2, 4, 8 cores) with three slave nodes using the 150M dataset. Figure 4 [Figure 4: see original paper] shows runtime versus core count.

Except for SVM (which first decreased then increased), runtime decreased as core count increased, demonstrating good scalability. However, the decreasing rate was non-linear, with 4 cores showing higher resource utilization than 8 cores. NB performed best with runtimes of 33.14s, 19.59s, and 12.12s.

Experiment 4 evaluated scalability by varying slave node counts (3-7 nodes) with 4-core nodes using the 150M dataset. Figure 5 [Figure 5: see original paper] shows runtime versus node count.

Runtimes decreased as node count increased up to 5 nodes, after which improvement slowed, indicating inter-node communication overhead constrains scalability. NB again performed best with runtimes of 19.59s, 14.92s, 12.52s, 11.31s, and 9.84s.

Experiment 5 evaluated scalability across data scales (15M to 300M) comparing single-node (4-core/4GB) and cluster (5 slave nodes, 4-core/4GB each) Spark environments. Table 2 and Figures 6-7 [Figure 6: see original paper][Figure 7: see original paper] show results.

Single-node runtime increased proportionally with data scale, with rapid acceleration beyond 75M. Cluster runtime also increased proportionally (except for SVM). NB showed the best performance with minimal runtime increase. Figure 8 [Figure 8: see original paper] shows speedup ratios increasing with data scale, with LR and SVM speedup <1 for datasets <75M, indicating no computational advantage for small data.

4.3 Discussion This paper proposes parallel versions of DT, LR, NB, RF, and SVM algorithms and applies them to big data sentiment analysis. Several findings warrant discussion:

First, algorithm efficiency differs significantly across runtime environments. Spark clusters offer clear advantages for massive sentiment classification, with

benefits increasing at larger scales. This necessitates Spark cluster adoption for big data sentiment analysis to save time and improve efficiency.

Second, algorithm effectiveness (accuracy, recall, F-value) also differs across environments. Contrary to expectations, Spark cluster modes do not significantly improve these metrics compared to traditional Sklearn. Most algorithms show decreased performance when moving from Sklearn to Spark, indicating substantial work remains to maintain effectiveness in Spark environments.

Third, cluster efficiency changes with node and core counts. While increased resources improve efficiency, the improvement rate is non-linear. Beyond certain thresholds, efficiency gains diminish. In our empirical study with the given data volumes, 4 cores and 5 nodes provided optimal efficiency with minimal overhead. This suggests that more nodes/cores are not always better; matching resource configuration to data scale thresholds maximizes resource savings.

Fourth, below certain dataset scales, parallel speedup ratios fall below 1, meaning clusters provide no advantage. In our study, most algorithms showed speedup <1 for datasets <75M, indicating traditional Sklearn is preferable for smaller datasets.

Overall, while this scale adaptation research focuses on text sentiment analysis, conclusions extend to related large-scale text processing domains including text classification, product recommendation, information retrieval, and data mining. However, Spark's limited data partitioning capability can cause load imbalance when task distribution is uneven, making it unsuitable for all data scales, types, and algorithms. Future work will examine larger data scales, more data types, and additional algorithms to further explore parallel task allocation and algorithm effectiveness under scale adaptation.

References

[1] BALTAS A, KANAVOS A, TSAKALIDIS A K. An Apache Spark implementation for sentiment analysis on Twitter data[C]//Proceedings of algorithmic aspects of cloud computing. Cham: Springer, 2016: 15-25.
[2] MING Junren. Research on text sentiment analysis algorithm integrating semantic association mining[J]. Library and Information Service, 2012, 56(15): 99-103.
[3] TANG Xiaobo, LAN Yuting. Microblog product review sentiment analysis based on feature ontology[J]. Library and Information Service, 2016, 60(16): 121-127.
[4] XU H, ZHANG F, WANG W. Implicit feature identification in Chinese reviews using explicit topic mining model[J]. Knowledge-based systems, 2015, 76(3): 166-175.
[5] LIU Wen, GAO Feng, HONG Lingzi. Disaster network public opinion research based on sentiment analysis: A case study of Ya'an earthquake[J]. Library and Information Service, 2013, 57(20): 104-110.
[6] YU Chuanming. Cross-domain text sentiment analysis based on deep

recurrent neural networks[J]. Library and Information Service, 2018, 62(11): 23-34.
[7] YU Chuanming, FENG Bolin, AN Lu. Cross-domain sentiment analysis based on deep representation learning[J]. Data Analysis and Knowledge Discovery, 2017(7): 73-81.
[8] YU Chuanming, FENG Bolin, TIAN Xin, et al. Multilingual text sentiment analysis based on deep representation learning[J]. Journal of Shandong University: Natural Science Edition, 2018, 53(3): 13-23.
[9] YU Chuanming, AN Lu. From small data to big data: Three challenges facing opinion retrieval[J]. Information Studies: Theory & Application, 2016, 39(2): 13-19.
[10] XIANG Xiaojun, GAO Yang, SHANG Lin, et al. Parallelization of massive text classification based on Hadoop platform[J]. Computer Science, 2011, 38(10): 184-188.
[11] GLUSHKOVA D, JOVANOVIC P, ABELLO A. MapReduce performance model for Hadoop 2.x[EB/OL]. [2017-12-30]. https://doi.org/10.1016/j.is.2017.11.006.
[12] PAN J, HUAY, LIU X, et al. Bagging-based logistic regression with Spark: a medical data mining method[C]//International conference on advances in mechanical engineering and industrial informatics. Atlantis: Atlantis Press, 2016: 1553-1559.
[13] MOGHADDAM G, AHLAWAT K, SINGH A P. Performance analysis of machine learning techniques on big data using Apache Spark[C]//International conference on recent developments in science, engineering and technology. Singapore: Springer, 2017: 17-26.
[14] GUO Shunli, ZHANG Xiangxian. Research on construction method of sentiment dictionary for Chinese book reviews[J]. New Technology of Library and Information Service, 2016, 32(2): 67-74.
[15] MOGHADDAM S, ESTER M. Opinion digger: an unsupervised opinion miner from unstructured product reviews[C]//ACM international conference on information and knowledge management. New York: ACM, 2010: 1825-1828.
[16] LIU Lizhen, ZHAO Xinlei, WANG Hanshi, et al. Domain sentiment ontology construction based on product features[J]. Transactions of Beijing Institute of Technology, 2015, 35(5): 538-544.
[17] WANG H, NIE X, LIU L, et al. A fuzzy domain sentiment ontology based opinion mining approach for Chinese online product reviews[J]. Journal of computers, 2013, 8(9): 2225-2231.
[18] ZHU J, WANG H, ZHU M, et al. Aspect-based opinion polling from customer reviews[J]. IEEE transactions on affective computing, 2011, 2(1): 37-49.
[19] YAN Z, XING M, ZHANG D, et al. EXPRS: an extended pagerank method for product feature extraction from online consumer reviews[J]. Information & management, 2015, 52(7): 850-858.
[20] PANG B, LEE L, VAITHYANATHAN S. Thumbs up?: sentiment classification using machine learning techniques[C]//Proceedings of the ACL-02 conference on empirical methods in natural language processing, 2002: 79-86.
[21] DAVIDOV D, TSUR O, RAPPOPORT A. Enhanced sentiment learning

using Twitter hashtags and smileys[C]//International conference on computational linguistics: posters. Stroudsburg: Association for Computational Linguistics, 2010: 241-249.
[22] SU Ying, ZHANG Yong, HU Po, et al. Sentiment analysis combining Naive Bayes and LDA[J]. Computer Applications, 2016, 36(6): 1613-1618.
[23] WANG S, MANNING C D. Baselines and bigrams: simple, good sentiment and topic classification[C]//Meeting of the Association for Computational Linguistics: short papers. Stroudsburg: Association for Computational Linguistics, 2012: 90-94.
[24] CHEN Zhao, XU Ruifeng, GUI Lin, et al. Chinese sentiment analysis combining convolutional neural networks and word sentiment sequence features[J]. Journal of Chinese Information Processing, 2015, 29(6): 172-182.
[25] FAN L, ZHANG Y, DANG Y, et al. Analyzing sentiments in Web 2.0 social media data in Chinese: experiments on business and marketing related Chinese Web forums[J]. Information technology & management, 2013, 14(3): 231-242.
[26] HE Yue, ZHU Tingting. Research on haze public opinion based on microblog sentiment analysis and social network analysis[J]. Information Science, 2018(7): 91-97.
[27] AN Lu, WU Lin. Evolution analysis of microblog public opinion for emergency events integrating topic and sentiment features[J]. Library and Information Service, 2017, 61(15): 120-129.
[28] YOU Liping, WANG Jiamin. Electronic commerce customer satisfaction measurement based on sentiment analysis and VIKOR multi-attribute decision method[J]. Journal of the China Society for Scientific and Technical Information, 2015, 34(10): 1098-1110.
[29] SHOU Huanrong, DENG Shuqing, XU Jian. Network rumor identification method based on sentiment analysis[J]. Data Analysis and Knowledge Discovery, 2017(7): 44-51.
[30] XIAO Lu, CHEN Guo, LIU Jiyun. Enterprise product-level competitor identification based on sentiment analysis using user reviews as data source[J]. Library and Information Service, 2016, 60(1): 83-90.
[31] ZHU Jizhao, JIA Yantao, XU Jun, et al. SparkCRF: a parallel CRFs algorithm implementation based on Spark[J]. Journal of Computer Research and Development, 2016, 53(8): 1819-1828.
[32] CHEN J, LI K, TANG Z, et al. A parallel random forest algorithm for big data in a Spark cloud computing environment[J]. IEEE transactions on parallel & distributed systems, 2017, 28(4): 919-933.
[33] HAI M, ZHANG Y, ZHANG Y. A performance evaluation of classification algorithms for big data[J]. Procedia computer science, 2017, 122(1): 1100-1107.
[34] SONG Jie, SUN Zongzhe, MAO Keming, et al. MapReduce big data processing platforms and algorithms: a survey[J]. Journal of Software, 2017, 28(3): 514-543.
[35] SALLOUM S, DAUTOV R, CHEN X, et al. Big data analytics on Apache Spark[J]. International journal of data science & analytics, 2016, 1(3/4): 145-164.

[36] XING Xiaoyu. Research on parallelization of decision tree classification algorithm and its application[D]. Kunming: Yunnan University of Finance and Economics, 2010.
[37] WEI Jie. Research on Bayesian text classification learning under MapReduce framework[D]. Taiyuan: Shanxi University of Finance and Economics, 2012.
[38] LUO Yuanshuai. Research on parallel text classification algorithm based on random forest and Spark[D]. Chengdu: Southwest Jiaotong University, 2016.
[39] LIU Zeqian, PAN Zhisong. Research on parallel SVM algorithm based on Spark[J]. Computer Science, 2016, 43(5): 238-242.
[40] THINKNOOK. Twitter sentiment analysis training corpus (dataset)[EB/OL]. [2017-12-30]. http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/.
[41] GO A, BHAYANI R, HUANG L. Sentiment140[EB/OL]. [2017-12-30]. https://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip.
[42] HEREDIA B, KHOSHGOFTAAR T M, PRUSA J, et al. Cross-domain sentiment analysis: an empirical investigation[C]//International conference on information reuse and integration. New York: IEEE, 2016: 160-165.
[43] GOEL A, GAUTAM J, KUMAR S. Real time sentiment analysis of Tweets using Naive Bayes[C]//International conference on next generation computing technologies. New York: IEEE, 2016: 257-260.
[44] LIMA M L, NASCIMENTO T P, LABIDI S, et al. Using sentiment analysis for stock exchange prediction[J]. International journal of artificial intelligence & applications, 2016, 7(1): 59-67.
[45] FRIEDRICH N, BOWMAN T D, STOCK W G, et al. Adapting sentiment analysis for tweets linking to scientific papers[EB/OL]. [2017-12-30]. http://cn.arxiv.org/pdf/1507.01967v1.

Author Contributions

Yu Chuanming: Conceptualization, data acquisition, initial draft writing, revision
Yuan Sai: Machine learning comparative experiments, initial draft writing, revision
Wang Feng: Big data platform construction, revision
An Lu: Conceptualization, revision

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