MR-SAT: A MapReduce Algorithm for Big Data Sentiment Analysis on Twitter

Nikolaos Nodarakis[1], Spyros Sioutas[2], Athanasios K. Tsakalidis[1] and Giannis Tzimas[3]

[1]*Computer Engineering and Informatics Department, University of Patras, 26504 Patras, Greece*

[2]*Department of Informatics, Ionian University, 49100 Gorfu, Greece*

[3]*Computer & Informatics Engineering Department, Technological Educational Institute of Western Greece, 26334 Patras, Greece*

Abstract: Sentiment analysis on Twitter data has attracted much attention recently. People tend to express their feelings freely, which makes Twitter an ideal source for accumulating a vast amount of opinions towards a wide diversity of topics. In this paper, we develop a novel method to harvest sentiment knowledge in the MapReduce framework. Our algorithm exploits the hashtags and emoticons inside a tweet, as sentiment labels, and proceeds to a classification procedure of diverse sentiment types in a parallel and distributed manner. Moreover, we utilize Bloom filters to compact the storage size of intermediate data and boost the performance of our algorithm. Through an extensive experimental evaluation, we prove that our solution is efficient, robust and scalable and confirm the quality of our sentiment identification.

1 INTRODUCTION

Twitter is one of the most popular social network websites and launched in 2006. It is a wide spreading instant messaging platform and people use it to get informed about world news, videos that have become viral, etc. Inevitably, a cluster of different opinions, that carry rich sentiment information and concern a variety of entities or topics, is formed. Sentiment is defined as "A thought, view, or attitude, especially one based mainly on emotion instead of reason"[1] and describes someone's mood or judge towards a specific entity. User-generated content that captures sentiment information has proved to be valuable and its use is widespread among many internet applications and information systems, such as search engines.

Hashtags are a convention for adding additional context and metadata to tweets. They are created by users as a way to categorize their message and/or highlight a topic and are extensively utilized in tweets (Wang et al., 2011). Moreover, they provide the ability to people to search tweets that refer to a common subject. The creation of a hashtag is achieved by prefixing a word with a hash symbol (e.g. #love). Emoticon refers to a digital icon or a sequence of key-board symbols that serves to represent a facial expression, as :-) for a smiling face[2]. Both, hashtags and emoticons, provide a fine-grained sentiment learning at tweet level which makes them suitable to be leveraged for opinion mining.

Although the problem of sentiment analysis has been studied extensively during recent years, existing solutions suffer from certain limitations. One problem is that the majority of approaches is bounded in centralized environments. Moreover, sentiment analysis is based on, it terms of methodology, natural language processing techniques and machine learning approaches. However, this kind of techniques are time-consuming and spare many computational resources. Consequently, at most a few thousand records can be processed by such techniques without exceeding the capabilities of a single server (Agarwal et al., 2011; Davidov et al., 2010; Jiang et al., 2011; Wang et al., 2011). Since millions of tweets are published daily on Twitter, it is more than clear that underline solutions are not sufficient. Consequently, high scalable implementations are required in order to acquire a much better overview of sentiment tendency towards a topic.

In this paper, we propose MR-SAT: a novel

---

[1]http://www.thefreedictionary.com/sentiment

[2]http://dictionary.reference.com/browse/emoticon

MapReduce Algorithm for Big Data Sentiment Analysis on Twitter implemented in Hadoop (White, 2012), the open source MapReduce implementation (Dean and Ghemawat, 2004). Our algorithm exploits the hashtags and emoticons inside a tweet, as sentiment labels, in order to avoid the time-intensive manual annotation task. After that, we build the feature vectors of training and test set and proceed to a classification procedure in a fully distributed manner using an AkNN query. Additionally, we encode features using Bloom filters to compress the storage space of the feature vectors. Through an extensive experimental evaluation we prove that our solution is efficient, robust and scalable and confirm the quality of our sentiment identification.

The rest of the paper is organized as follows: in Section 2 we discuss related work and in Section 3 we present how our algorithm works. After that, we proceed to the experimental evaluation of our approach in Section 4, while in Section 5 we conclude the paper and present future steps.

2 RELATED WORK

Early opinion mining studies focus on document level sentiment analysis concerning movie or product reviews (Hu and Liu, 2004; Zhuang et al., 2006) and posts published on webpages or blogs (Zhang et al., 2007). Respectively, many efforts have been made towards the sentence level sentiment analysis (Wilson et al., 2009; Yu and Hatzivassiloglou, 2003) which examines phrases and assigns to each one of them a sentiment polarity (positive, negative, neutral).

Many researchers confront the problem of sentiment analysis by applying machine learning approaches and/or natural language processing techniques. In (Pang et al., 2002), the authors employ three machine learning techniques to classify movie reviews as positive or negative. On the other hand, the authors in (Nasukawa and Yi, 2003) investigate the proper identification of semantic relationships between the sentiment expressions and the subject within online articles. Moreover, the method described in (Ding and Liu, 2007) proposes a set of linguistic rules together with a new opinion aggregation function to detect sentiment orientations in online product reviews.

Nowadays, Twitter has received much attention for sentiment analysis, as it provides a source of massive user-generated content that captures a wide aspect of published opinions. In (Barbosa and Feng, 2010), the authors propose a 2-step classifier that separates messages as subjective and objective, and fur-

ther distinguishes the subjective tweets as positive or negative. The approach in (Davidov et al., 2010) exploits the hashtags and smileys in tweets and evaluate the contribution of different features (e.g. unigrams) together with a kNN classifier. In this paper, we adopt this approach and create a parallel and distributed version of the algorithm for large scale Twitter data. A three-step classifier is proposed in (Jiang et al., 2011) that follows a target-dependent sentiment classification strategy. Moreover, the authors in (Wang et al., 2011) perform a topic sentiment analysis in Twitter data through a graph-based model. A more recent approach (Yamamoto et al., 2014), investigates the role of emoticons for multidimensional sentiment analysis of Twitter by constructing a sentiment and emoticon lexicon. A large scale solution is presented in (Khuc et al., 2012) where the authors build a sentiment lexicon and classify tweets using a MapReduce algorithm and a distributed database model. Although the classification performance is quite good, the construction of sentiment lexicon needs a lot of time. Our approach is much simpler and, to our best knowledge, we are the first to present a robust large scale approach for opinion mining on Twitter data without the need of building a sentiment lexicon or proceeding to any manual data annotation.

3 MR-SAT APPROACH

Assume a set of hashtags $H = \{h_1, h_2, \ldots, h_n\}$ and a set of emoticons $E = \{em_1, em_2, \ldots, em_m\}$ associated with a set of tweets $T = \{t_1, t_2, \ldots, t_l\}$ (training set). Each $t \in T$ carries only one sentiment label from $L = H \cup E$. This means that tweets containing more that one labels from L are not candidates for T, since their sentiment tendency may be vague. However, there is no limitation in the number of hashtags or emoticons a tweet can contain, as long as they are non-conflicting with L. Given a set of unlabelled tweets $TT = \{tt_1, tt_2, \ldots, tt_k\}$ (test set), we aim to infer the sentiment polarities $p = \{p_1, p_2, \ldots, p_k\}$ for TT, where $p_i \in L \cup \{neu\}$ and neu means that the tweet carries no sentiment information. We build a tweet-level classifier C and adopt a kNN strategy to decide the sentiment tendency $\forall tt \in TT$. We implement C by adapting an existing MapReduce classification algorithm based on AkNN queries (Nodarakis et al., 2014), as described in Subsection 3.3.

3.1 Feature Description

In this subsection, we present in detail the features used in order to build classifier C. For each tweet we

combine its features in one feature vector. We apply the features proposed in (Davidov et al., 2010) with some necessary modifications to avoid the production of an exceeding amount of calculations, thus boosting the running performance of our algorithm.

3.1.1 Word and N-Gram Features

We treat each word in a tweet as a binary feature. Respectively, we consider 2-5 consecutive words in a sentence as a binary n-gram feature. If f is a word or n-gram feature, then

$$w_f = \frac{N_f}{count(f)} \qquad (1)$$

is the weight of f in the feature vector, N_f is the number of times f appear in the tweet and $count(f)$ declares the count of f in the Twitter corpus. Consequently, rare words and n-grams have a higher weight than common words and have a greater effect on the classification task. Moreover, we consider sequences of two or more punctuation symbols as word features. Unlike what authors propose in (Davidov et al., 2010), we do not include the substituted meta-words for URLs, references and hashtags (URL, REF and TAG respectively) as word features (see and Section 4). Also, the common word RT, which means "retweet", does not constitute a feature. The reason for omission of these words from the feature list lies in the fact that they appear in the majority of tweets inside the dataset. So, their contribution as features is negligible, whilst they lead to a great computation burden during the classification task.

3.1.2 Pattern Features

This is the main feature type and we apply the pattern definitions given in (Davidov and Rappoport, 2006) for automated pattern extractions. We classify words into three categories: high-frequency words (HFWs), content words (CWs) and regular words (RWs). A word whose corpus frequency is more (less) than F_H (F_C) is considered to be a HFW (CW). The rest of the words are characterized as RWs. In addition, we treat as HFWs all consecutive sequences of punctuation characters as well as URL, REF, TAG and RT meta-words for pattern extraction. We define a pattern as an ordered sequence of HFWs and slots for content words. The upper bound for F_C is set to 1000 words per million and the lower bound for F_H is set to 10 words per million. Observe that the F_H and F_C bounds allow overlap between some HFWs and CWs. To address this issue, we follow a simple strategy as described next. Assume fr is the frequency of a word

in the corpus; if $fr \in \left(F_H, \frac{F_H + F_C}{2}\right)$ the word is classified as HFW, else if $fr \in \left[\frac{F_H + F_C}{2}, F_C\right)$ the word is classified as CW.

We seek for patterns containing 2-6 HFWs and 1-5 slots for CWs. Moreover, we require patterns to start and to end with a HFW, thus a minimal pattern is of the form [HFW][CW slot][HFW]. Additionally, we allow approximate pattern matching in order to enhance the classification performance. Approximate pattern matching is the same as exact matching, with the difference that an arbitrary number of RWs can be inserted between the pattern components. Since the patterns can be quite long and diverse, exact matches are not expected in a regular base. So, we permit approximate matching in order to avoid large sparse feature vectors. The weight w_p of a pattern feature p is defined as in Equation (1) in case of exact pattern matching and as

$$w_p = \frac{\alpha \cdot N_p}{count(p)} \qquad (2)$$

in case of approximate pattern matching, where $\alpha = 0.1$ in all experiments.

3.1.3 Punctuation Features

The last feature type is divided into five generic features as follows: 1) tweet length in words, 2) number of exclamation mark characters in the tweet, 3) number of question mark characters in the tweet, 4) number of quotes in the tweet and 5) number of capital/capitalized words in the tweet. The weight w_p of a punctuation feature p is defined as

$$w_p = \frac{N_p}{M_p \cdot (M_w + M_{ng} + M_{pa})/3} \qquad (3)$$

where N_p is the number of times feature p appears in the tweet, M_p is the maximal observed value of p in the twitter corpus and M_w, M_{ng}, M_{pa} declare the maximal values for word, n-gram and pattern feature groups, respectively. So, w_p is normalized by averaging the maximal weights of the other feature types.

3.2 Bloom Filter Integration

Bloom filters are data structures proposed by Bloom (Bloom, 1970) for checking element membership in any given set. A Bloom filter is a bit vector of length z, where initially all the bits are set to 0. We can map an element into the domain between 0 and $z - 1$ of the Bloom filter, using q independent hash functions $hf_1, hf_2, ..., hf_q$. In order to store each element e into

142

the Bloom filter, e is encoded using the q hash functions and all bits having index positions $hf_j(e)$ for $1 \leq j \leq q$ are set to 1.

Bloom filters are quite useful and they compress the storage space needed for the elements, as we can insert multiple objects inside a single Bloom filter. In the context of this work, we employ Bloom filters to transform our features to numbers, thus reducing the space needed to store our feature vectors. More precisely, instead of storing a feature we store the index positions in the Bloom filter that are set to 1. Nevertheless, it is obvious that the usage of Bloom filters may impose errors when checking for element membership, since two different elements may end up having exactly the same bits set to 1. The error probability is decreased as the number of bits and hash functions used grows. As shown in the experimental evaluation, the side effects of Bloom filters are negligible and boost the performance of our algorithm.

3.3 kNN Classification Algorithm

In order to assign a sentiment label for each tweet in TT, we apply a kNN strategy. Initially, we build the feature vectors for all tweets inside the training and test datasets (F_T and F_{TT} respectively). Then, for each feature vector u in F_{TT} we find all the feature vectors in $V \subseteq F_T$ that share at least one word/n-gram/pattern feature with u (matching vectors). After that, we calculate the Euclidean distance $d(u,v), \forall v \in V$ and keep the k lowest values, thus forming $V_k \subseteq V$ and each $v_i \in V_k$ has an assigned sentiment label $L_i, 1 \leq i \leq k$. Finally, we assign u the label of the majority of vectors in V_k. If no matching vectors exist for u, we assign a "neutral" label. We build C by adjusting an already implemented AkNN classifier in MapReduce to meet the needs of opinion mining problem.

3.4 Algorithmic Description

In this subsection, we describe in detail the sentiment classification process as implemented in the Hadoop framework. We adjust an already implemented MapReduce AkNN classifier to meet the needs of opinion mining problem. Our approach consists of a series of four MapReduce jobs, with each job providing input to the next one in the chain. These MapReduce jobs are summarized in the following subsections[3].

---

[3]Pseudo-codes are available in a technical report in http://arxiv.org/abs/1602.01248

3.4.1 Feature Extraction

In this MapReduce job, we extract the features, as described in Subsection 3.1, of tweets in T and TT and calculate their weights. The output of the job is an inverted index, where the key is the feature itself and the value is a list of tweets that contain it.

The *Map* function takes as input the records from T and TT, extracts the features of tweets. Afterwards, for each feature it outputs a key-value record, where the feature itself is the key and the value consists of the id of the tweet, the class of the tweet and the number of times the feature appears inside the sentence. The *Reduce* function receives the key-value pairs from the Map function and calculates the weight of a feature in each sentence. Then, it forms a list l with the format $< t_1, w_1, c_1 : \ldots : t_x, w_x, c_x >$, where t_i is the id of the i-th tweet, w_i is the weight of the feature for this tweet and c_i is its class. For each key-value pair, the Reduce function outputs a record where the feature is the key and the value is list l.

3.4.2 Feature Vector Construction

In this step, we build the feature vectors F_T and F_{TT} needed for the subsequent distance computation process. To achieve this, we combine all features of a tweet into one single vector. Moreover, $\forall tt \in TT$ we generate a list (*training*) of tweets in T that share at least one word/n-gram/pattern feature.

Initially, the *Map* function separates for each feature f the tweets that contain it into two lists, *training* and *test* respectively. Also, for each f it outputs a key-value record, where the key is the tweet id that contains f and the value consists of f and weight of f. Next, $\forall v \in test$ it generates a record where the key is the id of v and the value is the *training* list. The *Reduce* function gathers key-value pairs with the same key and builds F_T and F_{TT}. For each tweet $t \in T$ ($tt \in TT$) it outputs a record where key is the id of t (tt) and the value is its feature vector (feature vector together with the *training* list).

3.4.3 Distance Computation

In MapReduce Job 3, we create pairs of matching vectors between F_T and F_{TT} and compute their Euclidean distance.

For each feature vector $u \in F_{TT}$, the *Map* function outputs all pairs of vectors v in *training* list of u. The output key-value record has as key the id of v and the value consists of the class of v, the id of u and the u itself. Moreover, the Map function outputs all feature vectors in F_T. The *Reduce* function concentrates $\forall v \in F_T$ all matching vectors in F_{TT} and computes

the Euclidean distances between pairs of vectors. The Reduce function produces key-value pairs where the key is the id of u and the value comprises of the id of v, its class and the Euclidean distance $d(u,v)$ between the vectors.

3.4.4 Sentiment Classification

This is the final step of our proposed approach. In this job, we aggregate for all feature vectors u in the test set, the k vectors with the lowest Euclidean distance to u, thus forming V_k. Then, we assign to u the label (class) $l \in L$ of the majority of V_k, or the *neu* label if $V_k = \emptyset$.

The *Map* function is very simple and it just dispatches the key-values pairs it receives to the *Reduce* function. For each feature vector u in the test set, the Reduce function keeps the k feature vectors with the lowest distance to u and then estimates the prevailing sentiment label l (if exists) among these vectors. Finally, it assigns to u the label l.

4 EXPERIMENTS

In this section, we conduct a series of experiments to evaluate the performance of our method under many different perspectives. Our cluster includes 4 computing nodes (VMs), each one of which has four 2.4 GHz CPU processors, 11.5 GB of memory, 45 GB hard disk and the nodes are connected by 1 gigabit Ethernet. On each node, we install Ubuntu 14.04 operating system, Java 1.7.0_51 with a 64-bit Server VM, and Hadoop 1.2.1.

We evaluate our method using two Twitter datasets (one for hashtags and one for emoticons) we have collected through the Twitter Search API[4] between November 2014 to August 2015. We have used four human non-biased judges to create a list of hashtags and a list emoticons that express strong sentiment (e.g #bored and :)). We performed some experimentation to exclude from the lists the hashtags and emoticons that either were abused by twitter users or returned a very small number of tweets. We ended up with a list of 13 hashtags and a list of 4 emoticons. We preprocessed the datasets we collected and kept only the English tweets which contained 5 or more proper English words[5] and do not contain two or more hashtags or emoticons from the aforementioned lists. Moreover, during preprocessing we have replaced URL links, hashtags and ref-

---

[4]https://dev.twitter.com/rest/public/search

[5]To identify the proper English word we used an available WN-based English dictionary

erences by URL/REF/TAG meta-words as stated in (Davidov et al., 2010). The final hashtags dataset contains 942188 tweets (72476 tweets for each class) and the final emoticons dataset contains 1337508 tweets (334377 tweets for each class). In both datasets, hashtags and emoticons are used as sentiment labels and for each sentiment label there is an equal amount of tweets. Finally, in order to produce no-sentiment datasets we used Sentiment140 API[6] (Go et al., 2009) and a dataset which is publicly available[7]. We fed the no hashtags/emoticons tweets contained in this dataset, to the Sentiment140 API and kept the set of neutral tweets. We produced two no-sentiment datasets by randomly sampling 72476 and 334377 tweets from the neutral dataset. These datasets are used for the binary classification experiments.

We assess the classification performance of our algorithm using the 10-fold cross validation method and measuring the harmonic f-score. For the Bloom filter construction we use 999 bits and 3 hash functions. In order to avoid a significant amount of computations that greatly affect the running performance of the algorithm, we define a weight threshold $w = 0.005$ for feature inclusion in the feature vectors. In essence, we eliminate the most frequent words that have no substantial contribution to the final outcome.

4.1 Classification Performance

In this subsection we measure the classification performance of our solution using the harmonic f-score. We use two experimental settings, the multi-class classification and the binary classification settings. Under multi-class classification we attempt to assign a single label to each of vectors in the test set. In the binary classification experiments, we classified a sentence as either appropriate for a particular label or as not bearing any sentiment. As stated and in (Davidov et al., 2010), the binary classification is a useful application and can be used as a filter that extracts sentiment sentences from a corpus for further processing. We also test how the performance is affected with and without using Bloom filters. The value k for the kNN classifier is equal to 50. The results of the experiments are displayed in Table 1. In case of binary classification, the results depict the average score for all classes.

For multi-class classification the results are not very good but still they are way above the random baseline. We also observe that the results with and without the Bloom filters are almost the same. Thus, we deduce that for multi-class classification the

---

[6]http://help.sentiment140.com/api

[7]https://archive.org/details/twitter_cikm_2010

Table 1: Classification results for emoticons and hashtags (BF stands for Bloom filter and NBF for no Bloom filter).

Setup	BF	NBF	Random baseline
Multi-class Hashtags	0.32	0.33	0.08
Multi-class Emoticons	0.55	0.56	0.25
Binary Hashtags	0.74	0.53	0.5
Binary Emoticons	0.77	0.69	0.5

Table 2: Fraction of tweets with no matching vectors.

Setup	BF	NBF
Multi-class Hashtags	0.05	0.01
Multi-class Emoticons	0.05	0.02
Binary Hashtags	0.05	0.03
Binary Emoticons	0.08	0.06

Table 3: Effect of k in classification performance.

Setup	$k = 50$	$k = 100$	$k = 150$	$k = 200$
Multi-class Hashtags BF	0.32	0.32	0.32	0.32
Multi-class Hashtags NBF	0.33	0.35	0.37	0.37
Multi-class Emoticons BF	0.55	0.55	0.55	0.55
Multi-class Emoticons NBF	0.56	0.58	0.6	0.6
Binary Hashtags BF	0.74	0.75	0.75	0.75
Binary Hashtags NBF	0.53	0.62	0.68	0.72
Binary Emoticons BF	0.77	0.77	0.78	0.78
Binary Emoticons NBF	0.69	0.75	0.78	0.79

Figure 1: Space compression of feature vector.

Bloom filters marginally affect the classification performance. Furthermore, the outcome for emoticons is significantly better than hashtags which is expected due to the lower number of sentiment types. This behavior can also be explained by the ambiguity of hashtags and some overlap of sentiments. In case of binary classification there is a notable difference between the results with and without Bloom filters. These results may be somewhat unexpected but can be explicated when we take a look in Table 2. Table 2 presents the fraction of test set tweets that are classified as neutral because of the Bloom filters and/or the weight threshold w (no matching vectors are found). Notice that the integration of Bloom filters, leads to a bigger number of tweets with no matching vectors. Obviously, the excluded tweets have an immediate effect to the performance of the kNN classifier in case of binary classification. This happens since the number of tweets in the cross fold validation process is noticeably smaller compared to the multi-class classification. Overall, the results for binary classification with Bloom filters confirm the usefulness of our approach.

4.2 Effect of k

In this subsection, we attempt to alleviate the problem of low classification performance for binary classification without Bloom filters. To achieve this we measure the effect of k in the classification performance of the algorithm. We test four different configurations where $k \in \{50, 100, 150, 200\}$. The outcome of this experimental evaluation is demonstrated in Table 3. For both binary and multi-class classification, increasing k affects slightly (or not at all) the harmonic f-score when we embody Bloom filters. In the contrary (without Bloom filters), there is a great enhancement in the binary classification performance for hashtags and emoticons and a smaller improvement in case of multi-class classification. The inference of this experiment, is that larger values of k can provide a great
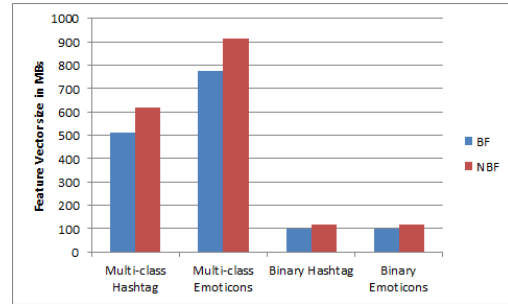
impulse in the performance of the algorithm when not using Bloom filters.

4.3 Space Compression

As stated and above, the Bloom filters can compact the space needed to store a set of elements, since more than one object can be stored to the bit vector. In this subsection, we elaborate on this aspect and present the compression ratio in the feature vectors when exploiting Bloom filters (in the way presented in Section 3.2) in our framework. The outcome of this measurement is depicted in Figure 1. In all cases, the Bloom filters manage to diminish the storage space required for the feature vectors by a fraction between 15-20%. According to the analysis made so far, the importance of Bloom filters in our solution is twofold. They manage to both preserve a good classification performance, despite any errors they impose, and compact the storage space of the feature vectors. Consequently, we deduce that Bloom filters are very beneficial, when dealing with large scale sentiment analysis data that generate an exceeding amount of features.

4.4 Running Time and Scalability

In this final experiment, we compare the running time for multi-class and binary classification and measure the scalability of our approach. Initially, we calculate the execution time in all cases in order to detect if the Bloom filters speedup or slow down the running performance of our algorithm. The results when $k = 50$ are presented in Figure 2. It is worth noted that in the

majority of cases, Bloom filters slightly boost the execution time performance. Despite needing more pre-processing time to produce the features with Bloom filters, in the end they pay off since the feature vector is smaller in size.
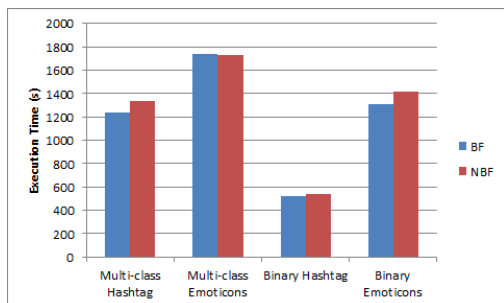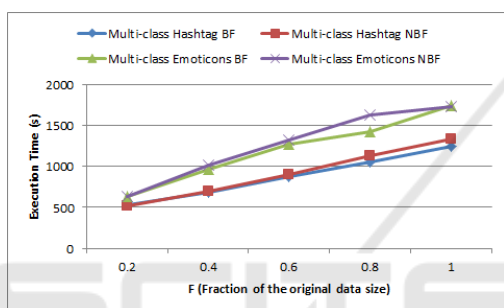

Figure 2: Running time.

Figure 3: Scalability.

Finally, we investigate the scalability of our approach. We test the scalability only for the multi-class classification case since the produced feature vector in much bigger compared to the binary classification case. We create new chunks smaller in size that are a fraction F of the original datasets, where $F \in \{0.2, 0.4, 0.6, 0.8\}$. Moreover, we set the value of k to 50. Figure 3 presents the scalability results of our approach. From the outcome, we deduce that our algorithm scales almost linearly as the data size increases in all cases. This proves that our solution is efficient, robust, scalable and therefore appropriate for big data sentiment analysis.

5 CONCLUSIONS AND FUTURE WORK

In the context of this work, we presented a novel method for sentiment learning in the MapReduce framework. Our algorithm exploits the hashtags and emoticons inside a tweet, as sentiment labels, and proceeds to a classification procedure of diverse sentiment types in a parallel and distributed manner. Also,

we utilize Bloom filters to compact the storage size of intermediate data and boost the performance of our algorithm. Through an extensive experimental evaluation, we prove that our system is efficient, robust and scalable.

In the near future, we plan to extend and improve our framework by exploring more features that may be added in the feature vector and will increase the classification performance. Furthermore, we wish to explore more strategies for F_H and F_C bounds in order to achieve better separation between the HFWs and CWs. Finally, we plan to implement our solution in other platforms (e.g. Spark) and compare the performance with the current implementation as well as other existing solutions, such Naive Bayes or Support Vector Machines.

REFERENCES

Agarwal, A., Xie, B., Vovsha, I., Rambow, O., and Passonneau, R. (2011). Sentiment analysis of twitter data. In *Proceedings of the Workshop on Languages in Social Media*, pages 30–38.

Barbosa, L. and Feng, J. (2010). Robust sentiment detection on twitter from biased and noisy data. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 36–44.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426.

Davidov, D. and Rappoport, A. (2006). Efficient unsupervised discovery of word categories using symmetric patterns and high frequency words. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 297–304.

Davidov, D., Tsur, O., and Rappoport, A. (2010). Enhanced sentiment learning using twitter hashtags and smileys. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 241–249.

Dean, J. and Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, pages 137–150.

Ding, X. and Liu, B. (2007). The utility of linguistic rules in opinion mining. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 811–812.

Go, A., Bhayani, R., and Huang, L. (2009). Twitter sentiment classification using distant supervision. *Processing*, pages 1–6.

Hu, M. and Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of the Tenth*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 168–177.

Jiang, L., Yu, M., Zhou, M., Liu, X., and Zhao, T. (2011). Target-dependent twitter sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, pages 151–160.

Khuc, V. N., Shivade, C., Ramnath, R., and Ramanathan, J. (2012). Towards building large-scale distributed systems for twitter sentiment analysis. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 459–464.

Nasukawa, T. and Yi, J. (2003). Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2Nd International Conference on Knowledge Capture*, pages 70–77.

Nodarakis, N., Pitoura, E., Sioutas, S., Tsakalidis, A. K., Tsoumakos, D., and Tzimas, G. (2014). Efficient multidimensional aknn query processing in the cloud. In *Database and Expert Systems Applications - 25th International Conference, DEXA 2014, Munich, Germany, September 1-4, 2014. Proceedings, Part I*, pages 477–491.

Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, pages 79–86.

Wang, X., Wei, F., Liu, X., Zhou, M., and Zhang, M. (2011). Topic sentiment analysis in twitter: A graph-based hashtag sentiment classification approach. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 1031–1040.

White, T. (2012). *Hadoop: The Definitive Guide, 3rd Edition*. O'Reilly Media / Yahoo Press.

Wilson, T., Wiebe, J., and Hoffmann, P. (2009). Recognizing contextual polarity: An exploration of features for phrase-level sentiment analysis. *Comput. Linguist.*, 35(3):399–433.

Yamamoto, Y., Kumamoto, T., and Nadamoto, A. (2014). Role of emoticons for multidimensional sentiment analysis of twitter. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, pages 107–115.

Yu, H. and Hatzivassiloglou, V. (2003). Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 129–136.

Zhang, W., Yu, C., and Meng, W. (2007). Opinion retrieval from blogs. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, pages 831–840.

Zhuang, L., Jing, F., and Zhu, X.-Y. (2006). Movie review mining and summarization. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 43–50.