Large Scale Sentiment Analysis on Twitter with Spark

Nikolaos Nodarakis
Computer Engineering and Informatics Department,
University of Patras
26504 Patras, Greece
nodarakis@ceid.upatras.gr

Spyros Sioutas
Department of Informatics,
Ionian University
49100 Corfu, Greece
sioutas@ionio.gr

Athanasios Tsakalidis
Computer Engineering and Informatics Department,
University of Patras
26504 Patras, Greece
tsak@ceid.upatras.gr

Giannis Tzimas
Computer & Informatics Engineering Department
Technological Educational Institute of Western Greece
26334 Patras, Greece
tzimas@cti.gr

ABSTRACT

Sentiment analysis on Twitter data has attracted much attention recently. One of the system's key features, is the immediacy in communication with other users in an easy, user-friendly and fast way. Consequently, 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. This amount of information offers huge potential and can be harnessed to receive the sentiment tendency towards these topics. However, since none can invest an infinite amount of time to read through these tweets, an automated decision making approach is necessary. Nevertheless, most existing solutions are limited in centralized environments only. Thus, they can only process at most a few thousand tweets. Such a sample, is not representative to define the sentiment polarity towards a topic due to the massive number of tweets published daily. In this paper, we go one step further and develop a novel method for sentiment learning in the Spark 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.

Keywords

big data; Bloom filters; classification; MapReduce; Spark; sentiment analysis; text mining; Twitter

1. INTRODUCTION

Nowadays, users tend to disseminate information, through short 140-character messages called "tweets", on different aspects on Twitter. Furthermore, they follow other users in order to receive their status updates. Naturally, Twitter constitutes a wide spreading instant messaging platform and people use it to get informed about world news, recent technological advancements, etc. Inevitably, a variety of opinion clusters that contain rich sentiment information 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.

Knowing the overall sentiment inclination towards a topic, may be proved extremely useful in certain cases. For instance, a technological company would like to know what their customers think about the latest product, in order to receive helpful feedback that will utilize in the production of the next device. So, it is obvious that an inclusive sentiment analysis for a time period after the release of the new product is needed. Moreover, user-generated content that captures sentiment information has proved to be valuable among many internet applications and information systems, such as search engines or recommendation systems.

In the context of this work, we utilize *hashtags* and *emoticons* as sentiment labels to perform classification of diverse sentiment types. Hashtags are a convention for adding additional context and metadata and are extensively utilized in tweets [23]. Their usage is twofold: they provide categorization of a message and/or highlight of a topic and they enhance the searching of tweets that refer to a common subject. A hashtag is created by prefixing a word with a hash symbol (e.g. #love). Emoticon refers to a digital icon or a sequence of keyboard symbols that serves to represent a facial expression, as `:-(` for a sad 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.

The problem of sentiment analysis has been studied extensively during recent years. The majority of existing so-

[1] `http://www.thefreedictionary.com/sentiment`
[2] `http://dictionary.reference.com/browse/emoticon`

lutions is bounded in centralized environments and base on natural language processing techniques and machine learning approaches. However, this kind of techniques are time-consuming and computationally intensive [16, 22]. As a result, it is prohibitive to process more than a few thousand tweets without exceeding the capabilities of a single server.

On the contrary, millions of tweets are published daily on Twitter. Consequently, underline solutions are neither sufficient nor suitable for opinion mining, since there is a huge mismatch between their processing capabilities and the exponential growth of available data [16]. It is more than clear that there is an imperative need to turn to high scalable solutions. Cloud computing technologies provide tools and infrastructure to create such solutions and manage the input data in a distributed way among multiple servers. The most prominent and notably efficient tool is the MapReduce [7] programming model, developed by Google, for processing large-scale data.

In this paper, we propose a novel distributed algorithm implemented in Spark [13, 21], an open source platform that translates the developed programs into MapReduce jobs. 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 perform a feature selection procedure to build the feature vectors of training and test set. Additionally, we embody Bloom filters to increase the performance of the algorithm. Finally, we adjust an existing MapReduce classification method based on AkNN queries to perform a fully distributed sentiment classification algorithm. We study various parameters that can affect the total computation cost and classification performance, such as size of dataset, number of nodes, increase of k, etc. by performing an extensive experimental evaluation. We prove that our solution is efficient, robust and scalable and verify the classification accuracy of our approach.

The rest of the paper is organized as follows: in Section 2 we discuss related work, the MapReduce model and Spark framework and in Section 3 we present how our algorithm works. More specifically, we explain how to build the feature vectors (for both the training and test dataset), we briefly describe the Bloom filter integration and display our Spark classification algorithm using pseudo-code. 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. PRELIMINARIES

2.1 Previous Work

Although the notion of sentiment analysis, or opinion mining, is relatively new, the research around this domain is quite extensive. Early studies focus on document level sentiment analysis concerning movie or product reviews [11, 30] and posts published on web pages or blogs [29]. Due to the complexity of document level opinion mining, many efforts have been made towards the sentence level sentiment analysis. The solutions presented in [25, 26, 28] examine phrases and assign to each one of them a sentiment polarity (positive, negative, neutral). A less investigated area is the topic-based sentiment analysis [15, 17] due to the difficulty to provide an adequate definition of topic and how to incorporate the sentiment factor into the opinion mining task.

The most common approaches to confront the problem of sentiment analysis include machine learning and/or natural language processing techniques. In [20], the authors employ Naive Bayes, Maximum Entropy and Support Vector Machines to classify movie reviews as positive or negative, and perform a comparison between the methods in terms of classification performance. On the other hand, Nasukawa and Yi [18] strive to identify semantic relationships between the sentiment expressions and the subject. Together with a syntactic parser and a sentiment lexicon their approach manages to augment the accuracy of sentiment analysis within web pages and online articles. Furthermore, Ding and Liu [8] define a set of linguistic rules together with a new opinion aggregation function to detect sentiment orientations in online product reviews.

During the last five years, Twitter has received much attention for sentiment analysis. In [2], the authors proceed to a 2-step classification process. In the first step, they separate messages as subjective and objective and in the second step they distinguish the subjective tweets as positive or negative. Davidov et al. [6] evaluate the contribution of different features (e.g. n-grams) together with a kNN classifier. They take advantage of the hashtags and smileys in tweets to define sentiment classes and to avoid manual annotation. In this paper, we adopt this approach and greatly extend it to support the analysis of large scale Twitter data. Agarwal et al. [1] investigate the use of a tree kernel model for detecting sentiment orientation in tweets. A three-step classifier is proposed in [12] that follows a target-dependent sentiment classification strategy. Moreover, a graph-based model is proposed in [23] to perform opinion mining in Twitter data from a topic-based perspective. A more recent approach [27], builds a sentiment and emoticon lexicon to support multidimensional sentiment analysis of Twitter data. A large scale solution is presented in [14] where the authors build a sentiment lexicon and classify tweets using a MapReduce algorithm and a distributed database model. Although the accuracy of the method is good, it suffers from the time-consuming construction of the sentiment lexicon. Our approach is much simpler and fully exploits the capabilities of Spark framework. To our best knowledge, we are the first to present a Spark-based large scale approach for opinion mining on Twitter data without the need of building a sentiment lexicon or proceeding to any manual data annotation.

2.2 MapReduce Model

Here, we briefly describe the MapReduce model [7]. The data processing in MapReduce is based on input data partitioning; the partitioned data is executed by a number of tasks in many distributed nodes. There exist two major task categories called *Map* and *Reduce* respectively. Given input data, a Map function processes the data and outputs key-value pairs. Based on the Shuffle process, key-value pairs are grouped and then each group is sent to the corresponding Reduce task. A user can define his own Map and Reduce functions depending on the purpose of his application. The input and output formats of these functions are simplified as key-value pairs. Using this generic interface, the user can solely focus on his own problem. He does not have to care how the program is executed over the distributed nodes, about fault tolerant issues, memory management, etc. The architecture of MapReduce model is depicted in Figure 1.

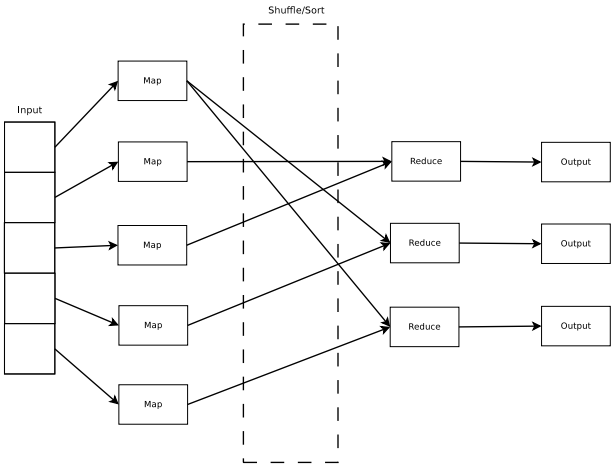

Figure 1: Architecture of MapReduce model

2.3 Spark Framework

Apache Spark [13, 21] is a fast and general engine for large-scale data processing. In essence, it is the evolution of Hadoop [10, 24] framework. Hadoop is the the open source implementation of the MapReduce model and is widely used for distributed processing among multiple servers. It is ideal for batch-based processes when we need to go through all data. However, its performance drops rapidly for certain problem types (e.g. when we have to deal with iterative or graph-based algorithms).

Spark is a unified stack of multiple closely integrated components and overcomes the issues of Hadoop. It has a Directed Acyclic Graph (DAG) execution engine that supports cyclic data flow and in-memory computing. As a result, it can ran programs up to 100x faster than Hadoop in memory, or 10x faster on disk. Spark includes a stack of libraries that combine SQL, streaming, machine learning and graph processing in a single engine. Spark offers many high level mechanisms, such as caching, and makes easy to build distributed applications in Java, Python, Scala and R. The applications are translated into MapReduce jobs and run in parallel. Furthermore, Spark can access different data sources, such as HDFS or HBase.

3. SENTIMENT CLASSIFICATION FRAMEWORK

In the beginning of this section, we define some notation used throughout this paper and then provide a formal definition of the confronted problem. After that, we introduce the features we use to build the feature vector. Finally, we describe our Spark algorithm using pseudo-code and proceed to a step by step explanation. Table 1 lists the symbols and their meanings.

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 [19], as described in Subsection 3.3.

Table 1: Symbols and their meanings

H	set of hashtags
E	set of emoticons
T	training set
TT	test set
L	set of sentiment labels of T
p	set of sentiment polarities of TT
C	AkNN classifier
w_f	weight of feature f
N_f	number of times feature f appears in a tweet
$count(f)$	count of feature f in corpus
fr_f	frequency of feature f in corpus
F_C	upper bound for content words
F_H	lower bound for high frequency words
M_f	maximal observed value of feature f in corpus
hf_i	i-th hash function
F_T	feature vector of T
F_{TT}	feature vector of TT
V	set of matching vectors

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 [6] with some necessary modifications. The reason of these alterations is to adapt the algorithm to the needs of large-scale processing in order to achieve an optimal performance.

3.1.1 Word and N-Gram Features

Each word in a tweet is treated as a binary feature. Respectively, a sequence of 2-5 consecutive words in a sentence is regarded as a binary n-gram feature. For each word or n-gram feature f we estimate its weight as $w_f = \frac{N_f}{count(f)}$. Consequently, rare words and n-grams have a higher weight than common words and have a greater effect on the classification task. Moreover, if we encounter sequences of two or more punctuation symbols inside a tweet, we consider them as word features. Unlike what authors propose in [6], 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). Additionally, 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

We apply the pattern definitions given in [5] for automated pattern extraction. The words are divided into three cate-

gories: high-frequency words (HFWs), content words (CWs) and regular words (RWs). Assume a word f and its corpus frequency fr_f; if $fr_f > F_H$, then f is considered to be a HFW. On the other hand, if $fr_f < F_C$, then f is considered to be a CW. The rest of the words are characterized as RWs. The word frequency is estimated from the training set rather than from an external corpus. 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, since they play an important role in pattern detection. 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. In contrary to [5], where F_H is set to 100 words per million, we provide a smaller lower bound since the experimental evaluation produced better results. 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: if $fr_f \in \left(F_H, \frac{F_H+F_C}{2}\right)$ the word is classified as HFW, else if $fr_f \in \left[\frac{F_H+F_C}{2}, F_C\right)$ the word is classified as CW. More strategies can be explored but this is out of the scope of this paper and is left for future work.

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 resembles 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 $w_p = \frac{N_p}{count(p)}$ in case of exact pattern matching and as $w_p = \frac{\alpha \cdot N_p}{count(p)}$ 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 $w_p = \frac{N_p}{M_p \cdot (M_w + M_{ng} + M_{pa})/3}$, where 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 [3] 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 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 are primary used to 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 bit vectors. In this way, we manage to boost the performance of our algorithm and slightly decrease the storage space needed for feature vectors. 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.

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 algorithm as implemented in the Spark framework. Our approach consists of a single Spark program that runs in parallel. The logical flow of our solution can be divided into four consecutive steps:

- **Feature Extraction:** Extract the features from all tweets in T and TT
- **Feature Vector Construction:** Build the feature vectors F_T and F_{TT} respectively
- **Distance Computation:** For each vector $u \in F_{TT}$ find the matching vectors (if any exist) in F_T
- **Sentiment Classification:** Assign a sentiment label $\forall tt \in TT$

The pseudo-code of our approach follows and we analyze each step in detail. Our algorithm receives as input the files containing the training and the test datasets. After reading the files, it creates a unified dataset in memory which is required for further processing.

At first, the algorithm utilizes the cached data and derives the aforementioned features using the function *GetFeatures()*. According to the feature type, it extracts for each tweet in the dataset the corresponding features. Then, it groups features by key and creates an inverted index. The feature plays the role of the key and the value is a list of tweets that contain the feature, along with the corresponding weight of the feature for each tweet. The union of all inverted indexes consists the feature vectors.

In the next step, for each feature we separate the tweets that contain it into two lists, LS_T (training set tweets) and

Sentiment Classification Algorithm

```
1: function SCA(training_file, test_file)
2:     t = GetContent(training_file);
3:     tt = GetContent(test_file);
4:     d = t.union(tt); // Create a united dataset
5:     d.cache(); // Cache dataset to memory

6:     // Get all feature types
7:     wf = GetFeatures(WORD, d);
8:     ngf = GetFeatures(NGRAM, d);
9:     pf = GetFeatures(PATTERN, d);
10:    puf = GetFeatures(PUNCTUATION, d);

11:    // Get feature vectors
12:    fv = wf.union(ngf).union(pf).union(puf);
13:    // Get matching vectors
14:    mvm = fv.flatMap(newMVMap());
15:    mvp = mvm.mapToPair(newMVPMap());
16:    mv = mvp.groupByKey();

17:    // Compute distances
18:    dcm = mv.flatMap(newDistCalcMap());
19:    dcpm = dcm.mapToPair(newDistCalcPMap());
20:    distCalc = dcpm.groupByKey();

21:    // Tweet classification
22:    tc = distCalc.mapValues(newMaxClass());
23:    // Calculate accuracy and return it
24:    j = tc.join(tt);
25:    f = j.filter((x, y) : x.class == y.class);
26:    returnf.count/tt.count;
27: end function

28: function GetFeatures(type, dataset)
29:    // Create feature objects based on type
30:    fm = dataset.flatMap(newRecordMap(type));
31:    // Map each object to a feature key
32:    fp = fm.mapToPair(newFeaturePairMap());
33:    // Find feature weight in each tweet
34:    f = fp.groupByKey().mapValues(newWeight());
35:    returnf.cache();
36: end function
```

LS_{TT} (test set tweets). After that, $\forall ltt \in LS_{TT}$ we create pairs for all elements in LS_T. As a result, $\forall tt \in TT$ we construct a set of matching vectors V. Using this set, we calculate the Euclidean distances and keep the k lowest values and form $V_k \subseteq V$. Finally, we assign $\forall tt \in TT$ the label of the majority of vectors in their respective V_k. The algorithm returns the classification accuracy of our method.

Our solution relies on high level operators offered by Spark. These operators (e.g. flatMap, mapToPair, etc.) are designed to distribute the workload equally among the nodes of the cluster to achieve high performance. Although our algorithm resembles a sequential execution of commands, it is fully distributed and exploits the capabilities offered by the framework.

4. EXPERIMENTAL EVALUATION

In this section, we conduct a series of experiments to evaluate the performance of our method under many different perspectives. More precisely, we take into consideration the effect of k and Bloom filters, the space compaction ratio, the size of the dataset and the number of nodes in the performance of our solution.

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 1.8.0_66 with a 64-bit Server VM, and Spark 1.4.1. One of the VMs serves as the master node and the other three VMs as the slave nodes. Moreover, we apply the following changes to the default Spark configurations: we use 12 total executor cores (4 for each slave machine), we set the executor memory equal to 8 GB and the driver memory to 4 GB.

We evaluate our method using two Twitter datasets (one for hashtags and one for emoticons) that we have collected through the Twitter Search API[3] 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 #amazed and `:(`). Then, we proceed to a cleaning task to exclude from the lists the hashtags and emoticons that either were abused by twitter users (e.g. #love) or returned a very small number of tweets. We ended up with a list of 13 hashtags (i.e. H = {#amazed, #awesome, #beautiful, #bored, #excited, #fun, #happy, #lol, #peace, #proud, #win, #wow, #wtf}) and a list of 4 emoticons (i.e. E = { `:)`, `:(`, `xD`, `<3`}).

We preprocessed the datasets we collected and kept only the English tweets which contained 5 or more proper English words[4] and do not contain two or more hashtags or emoticons from the aforementioned lists. Moreover, during preprocessing we have replaced URL links, hashtags and references by URL/REF/TAG meta-words as stated in [6]. 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). The size of the hashtags dataset is 102.78 MB and the size of the emoticons dataset is 146.4 MB. 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[5] [9] and the dataset used in [4], which is publicly available[6]. 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 (see Section 4.1).

We assess the classification performance of our algorithm using the 10-fold cross validation method and measuring the accuracy. 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.

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

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

[5] `http://help.sentiment140.com/api`

[6] `https://archive.org/details/twitter_cikm_2010`

Table 2: 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	0.37	0.35	0.08
Multi-class Emoticons	0.59	0.56	0.25
Binary Hashtags	0.73	0.71	0.5
Binary Emoticons	0.77	0.76	0.5

Table 3: 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

4.1 Classification Performance

In this subsection, we measure the classification performance of our solution using the classification accuracy. We define classification accuracy as $acc = |CT|/|TT|$, where $|CT|$ is the number of test set tweets that were classified correctly and $|TT|$ is the cardinality of TT. We present the results of two experimental configurations, the multi-class classification and the binary classification. Under multi-class classification setting, we attempt to assign a single sentiment label to each of vectors in the test set. In the binary classification experiment, we check if a sentence is suitable for a specific label or does not carry any sentiment inclination. As stated and in [6], the binary classification is a useful application and can be used as a filter that extracts sentiment sentences from a corpus for further processing. Moreover, we measure the influence of Bloom filters in the classification performance. The value k for the kNN classifier is set to 50. The results of the experiments are displayed in Table 2. In case of binary classification, the results depict the average score for all classes.

Looking at the outcome in Table 2 we observe that the performance of multi-class classification in not very good, although is way above the random baseline. Furthermore, the results with and without Bloom filters differ marginally. The same thing happens for the binary classification configuration, however this time the accuracy of our approach is notably better. This 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. Nevertheless, there is a slight increase in the classification performance of our algorithm when employing Bloom filters, which is somewhat unexpected. Table 3 presents the fraction of test set tweets that are classified as neutral because 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 slight effect to the performance of the kNN classifier. Overall, the results for binary classification verify the usefulness of our approach.

4.2 Effect of k

In this subsection, 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 4. For both binary and multi-class classification, increasing k affects slightly (or not at all) the classification accuracy when we embody Bloom filters. In the contrary ,without Bloom filters, there is a bigger enhancement in the accuracy performance for both classification configurations (up to 3%). The inference of this experiment, is that larger values of k can provide a good impulse in the performance of the algorithm when not using Bloom filters. However, larger values of k mean more processing time. Thus, Bloom filters manage to improve the binary classification performance of the algorithm and at the same time they reduce the total processing cost.

Table 4: Effect of k in classification performance

Setup	$k = 50$	$k = 100$	$k = 150$	$k = 200$
Multi-class Hashtags BF	0.37	0.37	0.37	0.38
Multi-class Hashtags NBF	0.35	0.36	0.37	0.38
Multi-class Emoticons BF	0.59	0.59	0.59	0.59
Multi-class Emoticons NBF	0.56	0.58	0.58	0.59
Binary Hashtags BF	0.73	0.73	0.73	0.74
Binary Hashtags NBF	0.71	0.72	0.73	0.74
Binary Emoticons BF	0.77	0.77	0.77	0.78
Binary Emoticons NBF	0.76	0.77	0.78	0.78

4.3 Space Compression

As stated and above, the Bloom filters can compact the space needed to store a set of elements. 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 Table 5. In the majority of the cases, the Bloom filters manage to marginally diminish the storage space required for the feature vectors (up to 3%). In one case (multi-class hashtags), the decrease in the required space is significant (almost 9%). The reasons for these small differences are two. First of all, in each Bloom filter we store only one feature (instead of more) because of the nature of our problem. Secondly, we keep in memory a Bloom filter object instead of a String object. But, the size that each object occupies in main memory is almost the same (Bloom filter is slightly smaller). Since the size of our input is not very big, we expect this gap to increase for larger datasets that will produce significantly more space-consuming feature vectors. Consequently, we deduce that Bloom filters can be very beneficial when dealing with large scale sentiment analysis data, that generate an exceeding amount of features during the feature vector construction step.

4.4 Running Time

In this experiment, we compare the running time for multi-class and binary classification. 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 all cases, Bloom filters slightly or greatly boost the execution time performance. Especially for the multi-class hashtags and binary emoticons cases, the level of time reduction reaches 17%. Despite needing more preprocessing time to produce the features with Bloom filters, in the end they pay off since the Spark operations work faster with Bloom filter objects. Moreover, observe that

Table 5: Space compression of feature vector

Setup	BF	NBF
Multi-class Hashtags	2338.8 MB	2553 MB
Multi-class Emoticons	3027.7 MB	3028 MB
Binary Hashtags	403.3 MB	404 MB
Binary Emoticons	1605.8 MB	1651.4 MB

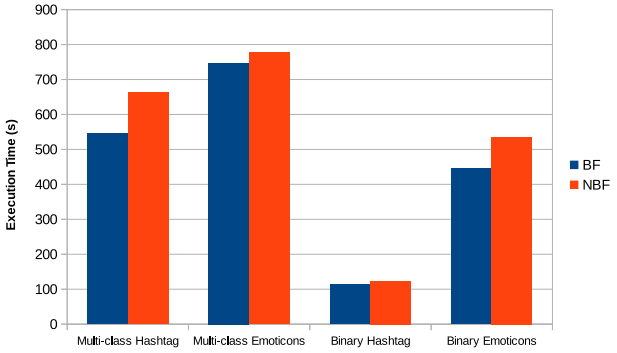

Figure 2: Running time

these configurations have the biggest compaction ratio according to Table 5. According to the analysis made so far, the importance of Bloom filters in our solution is threefold. They manage to preserve a good classification performance, despite any errors they impose, slightly compact the storage space of the feature vectors and enhance the running performance of our algorithm.

4.5 Scalability and Speedup

In this final experiment, we investigate the scalability and speedup of our approach. We test the scalability only for the emoticons dataset in the multi-class classification case, since it is the biggest dataset in MB. We create new chunks smaller in size that are a fraction F of the original dataset, 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 both cases.

Finally, we estimate the effect of the number of computing nodes. We test three different cluster configurations and the cluster consist of $N \in \{1, 2, 3\}$ slave nodes each time. Once again, we test the cluster configurations against the emoticons dataset in the multi-class classification case when $k = 50$. Figure 4 presents the speedup results of our approach. We observe that total running time of our solution tends to decrease as we add more nodes to the cluster. Due to the increment of number of computing nodes, the intermediate data are decomposed to more partitions that are processed in parallel. As a result, the amount of computations that undertakes each node decreases respectively.

The last two figures prove 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 Spark framework. Our algorithm exploits the hashtags and emoticons inside a tweet, as

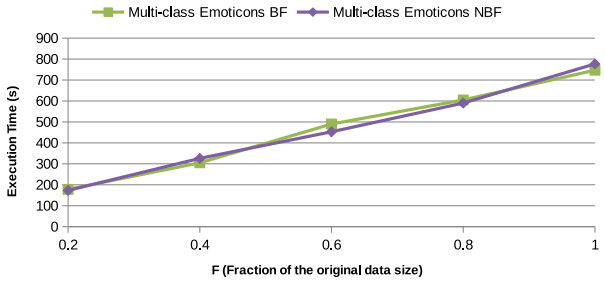

Figure 3: Scalability

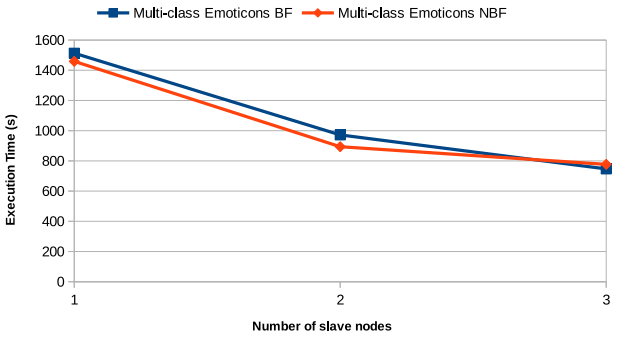

Figure 4: Speedup

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. Also, we schedule to investigate the effect of different Bloom filter bit vector sizes, in classification performance and storage space compression. Finally, we plan to compare the classification performance of our solution with other classification methods, such as Naive Bayes or Support Vector Machines.

6. REFERENCES

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

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

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

1970.

[4] Z. Cheng, J. Caverlee, and K. Lee. You are where you tweet: A content-based approach to geo-locating twitter users. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pages 759–768, 2010.

[5] D. Davidov and A. Rappoport. 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, 2006.

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

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

[8] X. Ding and B. Liu. 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, 2007.

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

[10] Hadoop. The apache software foundation: Hadoop homepage. `http://hadoop.apache.org/`, 2015. [Online; accessed 20-September-2015].

[11] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 168–177, 2004.

[12] L. Jiang, M. Yu, M. Zhou, X. Liu, and T. Zhao. 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, 2011.

[13] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia. *Learning Spark: Lightning-Fast Big Data Analysis*. O'Reilly Media, 2015.

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

[15] C. Lin and Y. He. Joint sentiment/topic model for sentiment analysis. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 375–384, 2009.

[16] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan and Claypool Publishers, 2010.

[17] Q. Mei, X. Ling, M. Wondra, H. Su, and C. Zhai. Topic sentiment mixture: Modeling facets and opinions in weblogs. In *Proceedings of the 16th International Conference on World Wide Web*, pages 171–180, 2007.

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

[19] N. Nodarakis, E. Pitoura, S. Sioutas, A. K. Tsakalidis, D. Tsoumakos, and G. Tzimas. kdann+: A rapid aknn classifier for big data. *T. Large-Scale Data- and Knowledge-Centered Systems*, 23:139–168, 2016.

[20] B. Pang, L. Lee, and S. Vaithyanathan. 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, 2002.

[21] Spark. The apache software foundation: Spark homepage. `http://spark.apache.org/`, 2015. [Online; accessed 27-December-2015].

[22] M. van Banerveld, N. Le-Khac, and M. T. Kechadi. Performance evaluation of a natural language processing approach applied in white collar crime investigation. In *Future Data and Security Engineering - First International Conference, FDSE 2014, Ho Chi Minh City, Vietnam, November 19-21, 2014, Proceedings*, pages 29–43, 2014.

[23] X. Wang, F. Wei, X. Liu, M. Zhou, and M. Zhang. 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, 2011.

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

[25] T. Wilson, J. Wiebe, and P. Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 347–354, 2005.

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

[27] Y. Yamamoto, T. Kumamoto, and A. Nadamoto. 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, 2014.

[28] H. Yu and V. Hatzivassiloglou. 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, 2003.

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

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