algorithms

MDPI

Article

Large Scale Implementations for Twitter Sentiment Classification

Andreas Kanavos [1,*], Nikolaos Nodarakis [1], Spyros Sioutas [2], Athanasios Tsakalidis [1], Dimitrios Tsolis [3] and Giannis Tzimas [4]

[1] Computer Engineering and Informatics Department, University of Patras, Patras 26504, Greece; nodarakis@ceid.upatras.gr (N.N.); tsak@ceid.upatras.gr (A.T.)
[2] Department of Informatics, Ionian University, Corfu 49100, Greece; sioutas@ionio.gr
[3] Department of Cultural Heritage Management and New Technologies, University of Patras, Agrinio 30100, Greece; dtsolis@upatras.gr
[4] Computer & Informatics Engineering Department, Technological Educational Institute of Western Greece, Patras 26334, Greece; tzimas@cti.gr
* Correspondence: kanavos@ceid.upatras.gr

Academic Editor: Bruno Carpentieri
Received: 8 December 2016; Accepted: 1 March 2017; Published: 4 March 2017

Abstract: Sentiment Analysis on Twitter Data is indeed a challenging problem due to the nature, diversity and volume of the data. People tend to express their feelings freely, which makes Twitter an ideal source for accumulating a vast amount of opinions towards a wide spectrum of topics. This amount of information offers huge potential and can be harnessed to receive the sentiment tendency towards these topics. However, since no one 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 in order to define the sentiment polarity towards a topic due to the massive number of tweets published daily. In this work, we develop two systems: the first in the MapReduce and the second in the Apache Spark framework for programming with Big Data. The algorithm exploits all hashtags and emoticons inside a tweet, as sentiment labels, and proceeds to a classification method of diverse sentiment types in a parallel and distributed manner. Moreover, the sentiment analysis tool is based on Machine Learning methodologies alongside Natural Language Processing techniques and utilizes Apache Spark's Machine learning library, MLlib. In order to address the nature of Big Data, we introduce some pre-processing steps for achieving better results in Sentiment Analysis as well as Bloom filters to compact the storage size of intermediate data and boost the performance of our algorithm. Finally, the proposed system was trained and validated with real data crawled by Twitter, and, through an extensive experimental evaluation, we prove that our solution is efficient, robust and scalable while confirming the quality of our sentiment identification.

Keywords: Apache Spark; Big Data; Bloom Filters; Hadoop; MapReduce; 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. Twitter constitutes a wide spreading instant messaging platform and people use it to get informed about world news, recent technological advancements, and so on. 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 prove 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 be utilized in the production of the next device. Therefore, it is obvious that an inclusive sentiment analysis for a time period after the release of a 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 putting together additional context and metadata and are extensively utilized in tweets [2]. 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, such as `:-(` for a sad face [3]. 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, and 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 terms of methodology, natural language processing techniques and machine learning approaches. However, these kinds of techniques are time-consuming and spare many computational resources [4,5]. Underlying 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 [4].

As a result, it is prohibitive to process more than a few thousand tweets without exceeding the capabilities of a single server [2,6–8]. It is more than clear that there is an imperative need to turn to highly 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 [9] programming model, developed by Google, (Googleplex, Mountain View, CA, USA) for processing large-scale data.

In this manuscript, we propose a novel distributed framework implemented in Hadoop [10], the open source MapReduce implementation [9] as well as in Spark [11], 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 sets. Additionally, we embody Bloom filters to increase the performance of the algorithm. Finally, we adjust an existing MapReduce classification method based on AkNN (all-(k)-nearest-neighbor) 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 manuscript is organized as follows: in Section 2, we discuss related work, as well as the Machine Learning Techniques implemented in the proposed work. In Section 3, MapReduce model and Spark framework are presented, while, in Section 4, the Sentiment Analysis Classification Framework is presented and in following the way that our algorithm works. More specifically, we explain how to build the feature vectors (for both the training and test dataset). Then, we briefly describe the Bloom filter integration and finally display the Sentiment Classification Algorithm using pseudo-code. Section 5 presents the steps of training as well as the two types of datasets for validating our framework. Moreover, Section 6 presents the evaluation experiments conducted and the results gathered. Ultimately, Section 7 presents conclusions and draws directions for future work.

2. Related Work

2.1. Sentiment Analysis and Classification Models

In the last decade, there has been an increasing interest in studies of Sentiment Analysis as well as emotional models. This is mainly due to the recent growth of data available in the World Wide Web, especially of those that reflect people's opinions, experiences and feelings [12]. Early opinion mining studies focus on document level sentiment analysis concerning movie or product reviews [13,14] and posts published on web pages or blogs [15].

Sentiment Analysis is studied in many different levels. In [16], authors implement an unsupervised learning algorithm that classifies reviews, thus performing document level classification. Due to the complexity of document level opinion mining, many efforts have been made towards the sentence level sentiment analysis. The solutions presented in [17–19] 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 [20,21] 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. Pang et al. [22] used Naive Bayes, Maximum Entropy and Support Vector Machines classifiers so as to analyze sentiment of movie reviews; they classify movie reviews as positive or negative, and perform a comparison between the methods in terms of classification performance. Boiy and Moens [23] utilized classification models with the aim of mining the sentiment out of multilingual web texts. On the other hand, the authors in [24] investigate the proper identification of semantic relationships between the sentiment expressions and the subject within online articles. 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, the method described in [25] defines 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 [26], tweets referring to Hollywood movies are analyzed. They focused on classifying the tweets and in following on analyzing the sentiment about Hollywood movies in different parts of the world. Other studies that investigate the role of emoticons on sentiment analysis of tweets are the ones in [27,28]. In both works, Lexicons of Emoticons are used to enhance the quality of the results. Authors in [29] propose a system that uses an SVM (Support Vector Machine) classifier alongside a rule-based classifier so as to improve the accuracy of the system. In [30], the authors proceed with a two-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.

There is a lot of research interest in studying different types of information dissemination processes on large graphs and social networks. Naveed et al. [31] analyze tweet posts and forecasts for a given post and the likelihood of being retweeted on its content. Authors indicate that tweets containing negative emoticons are more likely to be retweeted than tweets with positive emoticons. Agarwal et al. [6] investigate the use of a tree kernel model for detecting sentiment orientation in tweets. A three-step classifier is proposed in [8] that follows a target-dependent sentiment classification strategy. Moreover, a graph-based model is proposed in [2] 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.

In addition, several works in the SemEval competitions addressed the task of classifying the sentiment of tweets with hundreds of participants [32–35]. The evaluations are intended to explore the nature of meaning in language, as meaning is intuitive to humans and so transferring those intuitions to computational analysis has proved elusive. Moreover, other learning methods were implemented in Hadoop for classifying the polarity of tweets, e.g., the large-scale formulation of the Support Vector

Machine learning algorithm as presented in [36,37]. Another similar work is introduced in [38], where authors propose techniques to speed up the computation process for sentiment analysis. Specifically, they use tweet subjectivity in order to select the right training samples, and, in the following, they introduce the concept of EFWS (Effective Word Score) of a tweet that is derived from polarity scores of frequently used words, e.g., an additional heuristic that can be used to speed up the sentiment classification with standard machine learning algorithms. They achieve overall accuracies of around 80% for a training dataset of $100K$ tweets, a result very similar to our proposed manuscript.

Previous works regarding emotional content are the ones in [39,40]; they presented various approaches for the automatic analysis of tweets and the recognition of the emotional content of each tweet based on Ekman emotion model, where the existence of one or more out of the six basic human emotions (Anger, Disgust, Fear, Joy, Sadness and Surprise) is specified. Moreover, a cloud-based architecture was proposed in [41] where authors aim at creating a sentiment analysis tool for Twitter data based on Apache Spark cloud framework. The proposed system was trained and validated with real data crawled by Twitter and in following results are compared with the ones from real users. In addition, in [42], a novel method for Sentiment Learning in the Spark framework is presented; the proposed 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. The approach in [7] evaluates the contribution of different features (e.g., n-grams) together with a *k*NN classifier. Authors 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. A large-scale solution is presented in [43] 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.

2.2. Machine Learning Techniques

In the proposed manuscript, we utilized three classification algorithms in order to implement the Sentiment Analysis Tool. The three algorithms utilized are Naive Bayes, Logistic Regression and Decision Trees.

Naive Bayes is a simple multiclass classification algorithm based on the application of Bayes theorem. Each instance of the problem is represented as a feature vector, and it is assumed that the value of each feature is independent of the value of any other feature. One of the advantages of this algorithm is that it can be trained very efficiently, as it needs only a single pass to the training data. Initially, the conditional probability distribution of each feature given class is computed, and then Bayes theorem is applied to predict the class label of an instance.

Logistic Regression is a regression model where the dependent variable can take one out of a fixed number of values. It utilizes a logistic function to measure the relationship between the instance class and the features extracted from the input. Although widely used for binary classification, it can be extended to solve multiclass classification problems.

Decision Trees make up a classification algorithm that is based on a tree structure whose leaves represent class labels while branches represent combinations of features that result in the aforementioned classes. Essentially, it executes a recursive binary partitioning of the feature space. Each step is selected greedily, aiming for the optimal choice for the given step by maximizing the information gain.

3. Cloud Computing Preliminaries

3.1. MapReduce Model

MapReduce is a programming model that enables the process of large datasets on a cluster using a distributed and parallel algorithm [9]. 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. A MapReduce program consists of two main procedures, Map() and Reduce() respectively, and is executed in three steps: Map, Shuffle and Reduce. In the Map phase, input data is partitioned and each partition is given as an input to a worker that executes the map function. Each worker processes the data and outputs key-value pairs. In the Shuffle phase, key-value pairs are grouped by key and each group is sent to the corresponding Reducer.

A user can define their own Map and Reduce functions depending on the purpose of their 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. Apache Hadoop is a popular open source implementation of the Map Reduce model.

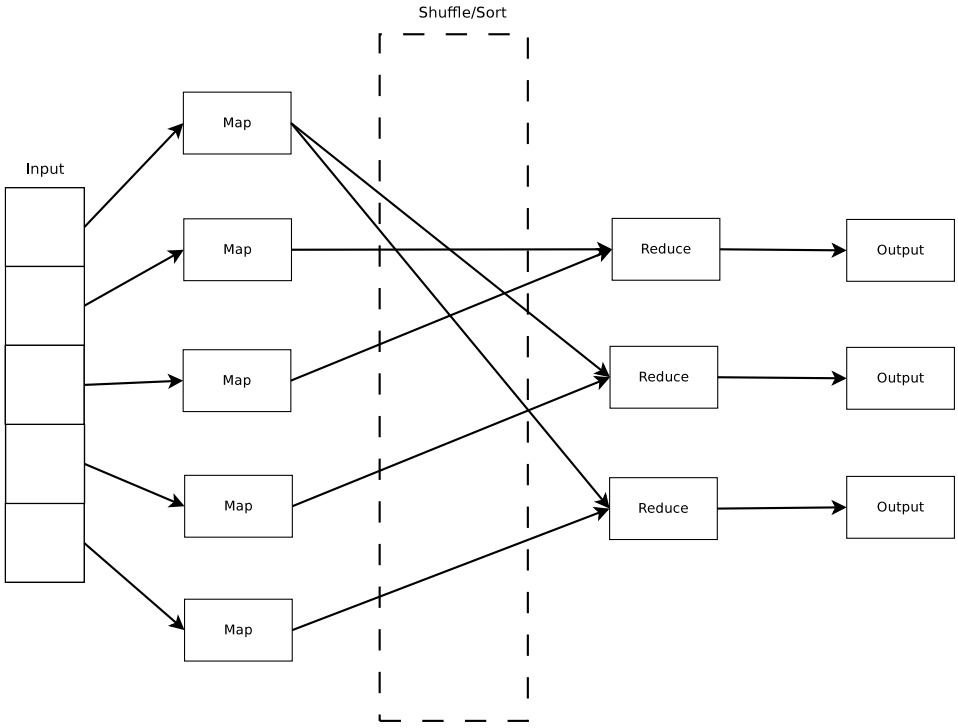

Figure 1. Architecture of MapReduce model.

3.2. Spark Framework

Apache Spark Framework [11,44] is a newer framework built in the same principles as Hadoop. While Hadoop is ideal for large batch processes, it drops in performance in certain scenarios, as in iterative or graph based algorithms. Another problem of Hadoop is that it does not cache intermediate data for faster performance, but, instead, it flushes the data to the disk between each step. In contrast, Spark maintains the data in the workers' memory, and, as a result, it outperforms Hadoop in algorithms that require many operations. It is a unified stack of multiple closely integrated components and overcomes the issues of Hadoop. In addition, it has a Directed Acyclic Graph (DAG) execution engine that supports cyclic data flow and in-memory computing. As a result, it can run programs up to $100x$ faster than Hadoop in memory, or $10x$ faster on disk. Spark offers APIs (Application Programming Interface) in Scala, Java, Python and R and can operate on Hadoop or standalone while using HDFS (Hadoop Distributed File System), Cassandra or HBase. The architecture of Apache Spark Framework is depicted in Figure 2.

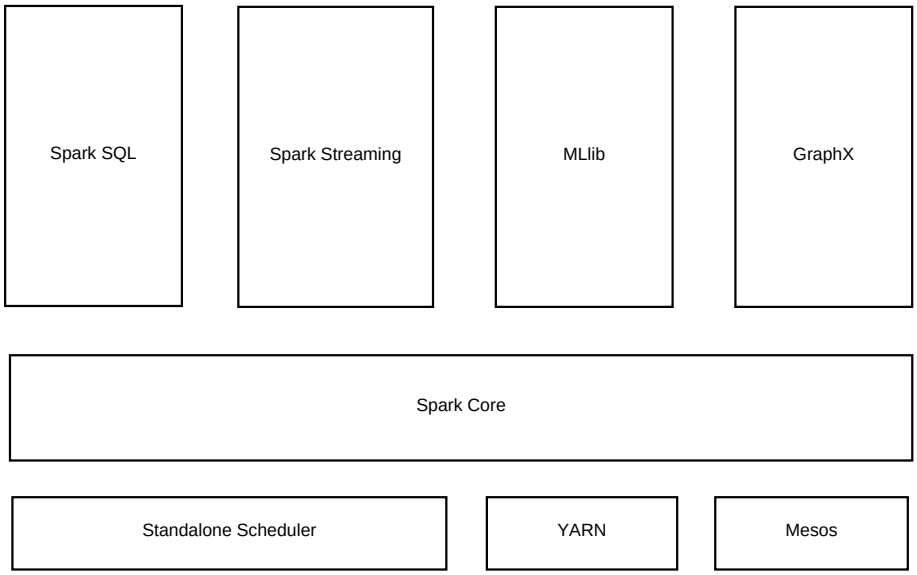

Figure 2. The Spark stack.

3.3. MLlib

Spark's ability to perform well on iterative algorithms makes it ideal for implementing Machine Learning Techniques as, at their vast majority, Machine Learning algorithms are based on iterative jobs. MLlib [45] is Apache Spark's scalable machine learning library and is developed as part of the Apache Spark Project. MLlib contains implementations of many algorithms and utilities for common Machine Learning techniques such as Clustering, Classification, and Regression.

4. Sentiment Analysis 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 that 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.

Table 1. Symbols and their meanings.

Symbol	Meaning
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

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 than one label from L are not candidates for T, since their sentiment tendency may be vague. However, there is no limitation on 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 [46], as described in Subsection 4.3.

4.1. 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 [7] with some necessary modifications. The reason for these alterations is to adapt the algorithm to the needs of large-scale processing in order to achieve an optimal performance.

4.1.1. 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. If f is a word or n-gram feature, then

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

is the weight of f in the feature vector, N_f is the number of times f appears 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, if we encounter sequences of two or more punctuation symbols inside a tweet, we consider them as word features. Unlike what authors propose in [7], 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. Therefore, their contribution as features is negligible, whilst they lead to a great computation burden during the classification task.

4.1.2. Pattern Features

We apply the pattern definitions given in [47] for automated pattern extraction. The words are divided into three categories: 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 [47], 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; otherwise, 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 an 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. Therefore, 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

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

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

4.1.3. 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

$$w_p = \frac{N_p}{M_p \cdot \left(M_w + M_{ng} + M_{pa}\right)/3}, \tag{3}$$

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. Therefore, w_p is normalized by averaging the maximal weights of the other feature types.

4.2. Bloom Filter Integration

Bloom filters are data structures proposed by Bloom [48] 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 into 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 lessened as the number of bits and hash functions used grows. As shown in the experimental evaluation, the side effects of Bloom filters are of minor importance.

4.3. 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 problems.

4.4. Algorithmic Description

In this subsection, we describe in detail the sentiment classification process as initially 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 Pseudo-codes are available in a technical report in [49].

Furthermore, as the next step in the specific subsection, we consider the implementation of the sentiment classification algorithm in the Spark framework. The approach consists of a single Spark program that runs in parallel. The logical flow of our solution can be divided, as previously, into the abovementioned 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 records provided as input to our algorithm have the format <tweet_id, class, text >, where `class` refers either to a sentiment label for tweets in T either to a no-sentiment flag for tweets in TT. In the following subsections, we describe each MapReduce job separately and analyze the Map and Reduce functions that take place in each one of them.

4.4.1. Feature Extraction

In this MapReduce job of Algorithm 1, 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. In the MapReduce Job 1 pseudo-code, we sum up the Map and Reduce functions of this process.

Algorithm 1: MapReduce Job 1

```
1:  Input: T and TT records
2:  function MAP(k1, v1)
3:      t_id = getId(v1); class = getClass(v1);
4:      features = getFeatures(v1);
5:      for all f ∈ features do // BF is BloomFilter
6:          output(BF(f.text), < t_id, f.count, class >);
7:      end for
8:  end function

9:  function REDUCE(k2, v2)
10:     feature_freq = 0;
11:     for all v ∈ v2 do
12:         feature_freq = feature_freq + v.count;
13:     end for
14:     l = List{};
15:     for all v ∈ v2 do
16:         weight = v.count / feature_freq;
17:         l.add(newRecord(v.t_id, weight, v.class));
18:     end for
19:     output(k2, l);
20: end function
```

The *Map* function takes as input the records from T and TT and 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

$< t_1, w_1, c_1 : ... : t_x, w_x, c_x >$, 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.

4.4.2. 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. The Map and Reduce functions are outlined in the following Algorithm 2.

Algorithm 2: MapReduce Job 2

```
1:  Input: Features F from tweets
2:  function MAP(k1, v1)
3:      f = getFeature(v1); t_list = getTweetList(v1);
4:      test = training = List{};
5:      for all t ∈ t_list do
6:          output(t.t_id, < f, t.weight >);
7:          if t.class ≠ NULL then
8:              training.add(newRecord(t.t_id, t.class));
9:          else
10:             test.add(newRecord(t.t_id, t.class));
11:         end if
12:     end for
13:     for all t ∈ test do
14:         output(t.t_id, training);
15:     end for
16: end function

17: function REDUCE(k2, v2)
18:     features = training = List{};
19:     for all v ∈ v2 do
20:         if v instanceOf List then
21:             training.addAll(v);
22:         else
23:             features.add(v);
24:         end if
25:     end for
26:     if training.size() > 0 then
27:         output(k2, < training, features >);
28:     else
29:         output(k2, features);
30:     end if
31: end function
```

Initially, the *Map* function separates $\forall f \in F$, the tweets that contain f into two lists—*training* and *test*, respectively. In addition, $\forall f \in F$ 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, $\forall v \in test$ 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 build 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).

4.4.3. Distance Computation

In Algorithm 3, we create pairs of matching vectors between F_T and F_{TT} and compute their Euclidean distance. The Map and Reduce functions are depicted in the pseudo-code that follows.

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 its 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 is comprised of the id of v, its class and the Euclidean distance $d(u, v)$ between the vectors.

Algorithm 3: MapReduce Job 3

Input: Feature Vectors F_T and F_{TT}
function MAP($k1, v1$)
    $t\_ids =$ **getTrainingIds**$(v1); v =$ **getVector**$(v1)$;
    $t\_id =$ **getId**$(v1)$;
    if $t\_ids.size() > 0$ **then**
        for all $u \in t\_ids$ **do**
            output$(u.t\_id, < u.class, t\_id, v >)$;
        end for
    else
        output(t_id, v);
    end if
end function

function REDUCE($k2, v2$)
    $ttv = List\{\}; tv = NULL$
    for all $v \in v2$ **do**
        if $v.class \neq NULL$ **then**
            $ttv.add(v)$;
        else
            $tv = v$;
        end if
    end for
    for all $tt \in ttv$ **do**
        ouput$(tt.t\_id, < tv.t\_id, tv.class, d(tt, tv) >)$;
    end for
end function

4.4.4. 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 Algorithm 4 is given below.

Algorithm 4: MapReduce Job 4

Input: Feature Vectors in the test set
function MAP($k1, v1$)
    $t\_id =$ **getTweetId**$(v1); val =$ **getValue**$(v1)$;
    output(t_id, val);
end function

function REDUCE($k2, v2$)
    $l\_k =$ **getKNN**$(v2)$;
    $H = HashMap < Class, Occurences > \{\}$;
    $H =$ **findClassOccur**(l_k);
    $max = 0; maxClass =$ **null**;
    for all $entry \in H$ **do**
        if $entry.occur > max$ **then**
            $max = entry.occur$;
            $maxClass = entry.class$;
        end if
    end for
    output$(k2, maxClass)$;
end function

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 v and then estimates the prevailing sentiment label l (if exists) among these vectors. Finally, it assigns the label l to u.

4.5. Preprocessing and Features

We examined both Binary and Ternary Classification on different datasets. On the Binary Classification case, we focus on the way that the dataset size affects the results, while in the Ternary Classification case, the focus is given on the impact of the different features of the feature vector given as an input to the classifier.

Regarding datasets we used for measuring our proposed algorithms' accuracy, a preprocessing step is utilized so as to discard all irrelevant data. Occurrences of usernames and URLs are replaced by special tags and each tweet is finally represented as a vector that consists of the following features:

- **Unigrams**, which are frequencies of words occurring in the tweets.
- **Bigrams**, which are frequencies of sequences of two words occurring in the tweets.
- **Trigrams**, which are frequencies of sequences of three words occurring in the tweets.
- **Username**, which is a binary flag that represents the existence of a user mention in the tweet.
- **Hashtag**, which is a binary flag that represents the existence of a hashtag in the tweet.
- **URL**, which is a binary flag that represents the existence of a URL in the tweet.
- **POS Tags**, where we used the Stanford NLT MaxEnt Tagger [50] to tag the tokenized tweets and the following are counted:

  1. Number of Adjectives,
  2. Number of Verbs,
  3. Number of Nouns,
  4. Number of Adverbs,
  5. Number of Interjections.

5. Implementation

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 four 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 an Ubuntu 14.04 operating system (Canonical Ltd., London, UK), Java 1.8.0_66 with a 64-bit Server VM, as well as Hadoop 1.2.1 and Spark 1.4.1 (for the different outcomes). 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 (four for each slave machine), and we set the executor memory equal to 8 GB and the driver memory to 4 GB.

5.1. Our Datasets for Evaluating MapReduce versus Spark Framework

We evaluate our method using two Twitter datasets (one for hashtags and one for emoticons) that we have collected through the Twitter Search API [51] 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 four emoticons (i.e., $E = \{$ `:)`, `:(`, `xD`, `<3`$\}$).

We preprocessed the datasets that we collected and kept only the English tweets which contained five or more proper English words. To identify the proper English word, we used an available WN-based English dictionary and do not include 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 [7]. The final hashtags dataset contains 942,188 tweets (72,476 tweets for each class) and the final emoticons dataset contains 1,337,508 tweets (334,377 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 non-sentiment datasets, we used the Sentiment140 API [52,53] and the dataset used in [54], which is publicly available [55]. We fed the no hashtags/emoticons tweets contained in this dataset into the

Sentiment140 API and kept the set of neutral tweets. We produced two non-sentiment datasets by randomly sampling 72.476 and 334.377 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 harmonic f-score. For the Bloom filter construction, we use 999 bits and three 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.

5.2. Open Datasets for Evaluating Machine Learning Techniques in Spark Framework

5.2.1. Binary Classification

For the Binary Classification, we used a dataset [56] of $1,578,627$ pre-classified tweets as Positive or Negative. We split the original dataset into segments of 1000, 2000, 5000, 10,000, 15,000, 20,000 and 25,000 tweets. Then, for each segment, all metadata were discarded and each tweet was transformed to a vector of unigrams; unigrams are the frequencies of each word in the tweets.

5.2.2. Ternary Classification

Regarding Ternary Classification, we used two datasets [57] that were merged into one, which eventually consists of 12,500 tweets. In the original datasets, each row contains the tweet itself, the sentiment, and other metadata related to the corresponding tweet. During the preprocessing, all irrelevant data were discarded, and we only used the actual text of the tweet, as well as the label that represents the sentiment, positive, negative or neutral. Each tweet is then tokenized and processed. Then, the ratios of the aforementioned numbers to the total number of tokens of each tweet are computed.

6. Results and Evaluation

6.1. Our Datasets for Evaluating MapReduce versus Spark Framework

6.1.1. 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 the multi-class classification setting, we attempt to assign a single sentiment label to each of the 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 [7], 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 the case of binary classification, the results depict the average score for all classes.

Table 2. Classification results for emoticons and hashtags (BF stands for Bloom filter and NBF for no Bloom filter).

Framework	**MapReduce**			**Spark**		
Setup	**BF**	**NBF**	**Random Baseline**	**BF**	**NBF**	**Random Baseline**
Binary Emoticons	0.77	0.69	0.5	0.77	0.76	0.5
Binary Hashtags	0.74	0.53	0.5	0.73	0.71	0.5
Multi-class Emoticons	0.55	0.56	0.25	0.59	0.56	0.25
Multi-class Hashtags	0.32	0.33	0.08	0.37	0.35	0.08

Looking at the outcome in Table 2, we observe that the performance of multi-class classification in not very good, despite being 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 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 the 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 3. Table 3 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 on the performance of the kNN classifier in the 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.

Table 3. Fraction of tweets with no matching vectors (BF for Bloom filter and NBF for no Bloom filter).

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

6.1.2. 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 4. For both binary and multi-class classification, increasing k affects slightly (or not at all) the harmonic f-score when we embody Bloom filters. On 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. 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 (BF for Bloom filter and NBF for no Bloom filter).

Framework	**MapReduce**				**Spark**			
Setup	$k = 50$	$k = 100$	$k = 150$	$k = 200$	$k = 50$	$k = 100$	$k = 150$	$k = 200$
Binary Emoticons BF	0.77	0.77	0.78	0.78	0.77	0.77	0.77	0.78
Binary Emoticons NBF	0.69	0.75	0.78	0.79	0.76	0.77	0.78	0.78
Binary Hashtags BF	0.74	0.75	0.75	0.75	0.73	0.73	0.73	0.74
Binary Hashtags NBF	0.53	0.62	0.68	0.72	0.71	0.72	0.73	0.74
Multi-class Emoticons BF	0.55	0.55	0.55	0.55	0.59	0.59	0.59	0.59
Multi-class Emoticons NBF	0.56	0.58	0.6	0.6	0.56	0.58	0.58	0.59
Multi-class Hashtags BF	0.32	0.32	0.32	0.32	0.37	0.37	0.37	0.38
Multi-class Hashtags NBF	0.33	0.35	0.37	0.37	0.35	0.36	0.37	0.38

6.1.3. 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 4.2) in our framework. The outcome of this measurement is depicted in Table 5.

Concerning MapReduce implementation, in all cases, the Bloom filters manage to minimize the storage space required for the feature vectors by a fraction between 15%–20%. On the other hand, for Spark implementation, the Bloom filters manage to marginally minimize the storage space required for the feature vectors (up to 3%) and, in one case (multi-class hashtags), the decrease in the required space is significant (almost 9%). 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.

There are two reasons for these small differences. 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 our minds a Bloom filter object instead of a String object. However, the size that each object occupies in the 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.

Table 5. Space compression of feature vectors (in MB).

Framework	**MapReduce**		**Spark**	
Setup	**BF**	**NBF**	**BF**	**NBF**
Binary Emoticons	98	116.76	1605.8	1651.4
Binary Hashtags	98	116.78	403.3	404
Multi-class Emoticons	776.45	913.62	3027.7	3028
Multi-class Hashtags	510.83	620.1	2338.8	2553

6.1.4. Running Time

In this final experiment, we compare the running time for multi-class and binary classification while also measuring the scalability of our approach. Initially, we calculate the execution time in all cases in order to detect if the Bloom filters speed up or slow down the running performance of our algorithm. The results when $k = 50$ are presented in Table 6 for MapReduce and Spark implementation. It is worth noting that, in the majority of cases, Bloom filters slightly 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 feature vector is smaller in size. Moreover,

observe that 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 probable errors, slightly compact the storage space of the feature vectors and enhance the running performance of our algorithm.

Table 6. Running time (in seconds).

Framework	**MapReduce**		**Spark**	
Setup	**BF**	**NBF**	**BF**	**NBF**
Binary Emoticons	1312	1413	445	536
Binary Hashtags	521	538	113	123
Multi-class Emoticons	1737	1727	747	777
Multi-class Hashtags	1240	1336	546	663

6.1.5. Scalability and Speedup

In this final experiment, we investigate the scalability and speedup of our approach. We test the scalability only for the multi-class classification case since the produced feature vector is 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 $F \in \{0.2, 0.4, 0.6, 0.8, 1\}$. Moreover, we set the value of k to 50. Table 7 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.

Table 7. Scalability (in seconds).

Framework	**MapReduce**					**Spark**				
1-1 Fraction F	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
Multi-class Emoticons BF	636	958	1268	1421	1737	178	305	490	605	747
Multi-class Emoticons NBF	632	1009	1323	1628	1727	173	326	453	590	777
Multi-class Hashtags BF	537	684	880	1058	1240	151	242	324	449	546
Multi-class Hashtags NBF	520	698	905	1135	1336	135	242	334	470	663

Finally, we estimate the effect of the number of computing nodes for Spark implementation. We test three different cluster configurations and the cluster consists 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 $k = 50$. Table 8 presents the speedup results of our approach. We observe that the total running time of our solution tends to decrease as we add more nodes to the cluster. Due to the increment of the 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 each node undertakes decreases respectively.

These results prove once again that our solution is efficient, robust, scalable and therefore appropriate for big data sentiment analysis.

Table 8. Speedup (in seconds).

Number of Slave Nodes	**1**	**2**	**3**
Multi-class Emoticons BF	1513	972	747
Multi-class Emoticons NBF	1459	894	777

6.2. Open Datasets for Evaluating Machine Learning Techniques in Spark Framework

The results of our work are presented in the following Tables 9–13. F-Measure is used as the evaluation metric of the different algorithms. For the binary classification problem (Table 9), we observe that Naive Bayes performs better than Logistic Regression and Decision Trees. It is also obvious that the dataset size plays a rather significant role for Naive Bayes, as the F-Measure value rises from 0.572 for a dataset of 1000 tweets to 0.725 for the dataset of 25,000 tweets. On the contrary, the performance of Logistic Regression and Decision Trees is not heavily affected by the amount of the tweets in the dataset.

Regarding ternary classification, Naive Bayes outperforms the other two algorithms as well, as it can be seen in Table 10, with Linear Regression following in the results. Interestingly, unigrams seem to be the feature that boosts the classification performance more than all of the other features that we examine, while the highest performance is observed for the vectors excluding trigrams. Moreover, the binary field representing the existence of a hashtag in the tweet affects the results, as, in all the experiments, the performance records smaller values without it. It can also be observed that all three algorithms perform better for positive and negative tweets than they do for neutral messages.

Table 9. F-Measure for Binary Classification for different dataset sizes.

Dataset Size	Decision Trees	Logistic Regression	Naive Bayes
1.000	0.597	0.662	0.572
5.000	0.556	0.665	0.684
10.000	0.568	0.649	0.7
15.000	0.575	0.665	0.71
20.000	0.59	0.651	0.728
25.000	0.56	0.655	0.725

Table 10. F-Measure for Ternary Classification for 12,500 tweets.

Classifier	Positive	Negative	Neutral	Total
Decision Trees	0.646	0.727	0.557	0.643
Logistic Regression	0.628	0.592	0.542	0.591
Naive Bayes	0.717	0.75	0.617	0.696

Table 11. F-Measure for Ternary Classification for Decision Trees for 12,500 tweets.

Features	Positive	Negative	Neutral	Total
Complete Feature Vector	0.646	0.727	0.557	0.643
w/o Unigrams	0.57	0.681	0.549	0.597
w/o Bigrams	0.647	0.729	0.557	0.644
w/o Trigrams	0.646	0.728	0.557	0.644
w/o User	0.646	0.727	0.557	0.643
w/o Hashtag	0.639	0.601	0.529	0.594
w/o URL	0.64	0.615	0.554	0.606
w/o POS Tags	0.659	0.729	0.56	0.65

Table 12. F-Measure for Ternary Classification for Logistic Regression for 12,500 tweets.

Features	Positive	Negative	Neutral	Total
Complete Feature Vector	0.628	0.592	0.542	0.591
w/o Unigrams	0.596	0.457	0.451	0.51
w/o Bigrams	0.616	0.6	0.546	0.59
w/o Trigrams	0.649	0.623	0.572	0.618
w/o User	0.625	0.6	0.54	0.592
w/o Hashtag	0.612	0.591	0.526	0.58
w/o URL	0.613	0.598	0.537	0.585
w/o POS Tags	0.646	0.585	0.512	0.587

Table 13. F-Measure for Ternary Classification for Naive Bayes for 12,500 tweets.

Features	Positive	Negative	Neutral	Total
Complete Feature Vector	0.717	0.75	0.617	0.696
w/o Unigrams	0.628	0.602	0.537	0.592
w/o Bigrams	0.714	0.769	0.629	0.705
w/o Trigrams	0.732	0.77	0.643	0.716
w/o User	0.718	0.751	0.618	0.698
w/o Hashtag	0.721	0.739	0.608	0.692
w/o URL	0.72	0.748	0.619	0.697
w/o POS Tags	0.716	0.748	0.617	0.695

7. Conclusions

In the context of this work, we have presented a tool that analyzes microblogging messages regarding their sentiment using machine learning techniques. This novel distributed framework was implemented in Hadoop as well as in Spark. 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. In addition, we utilize Bloom filters to compact the storage size of intermediate data and boost the performance of our algorithm. In addition, some classification algorithms are implemented in the Apache Spark cloud framework using Apache Spark's Machine Learning library, entitled MLlib. 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. One other future work will be the experimentation with different clusters so as to better evaluate Hadoop's and Spark's performance in regards to time and scalability. In addition, we plan to investigate the effect of different Bloom filter bit vector sizes, in classification performance and storage space compression. Moreover, we plan to compare the classification performance of our solution with other classification methods, such as Naive Bayes or Support Vector Machines.

Another future consideration is the adoption of aforementioned heuristics (e.g., the occurrence of emoticons) for handling complex semantic issues, such as irony that is typical of messages in Twitter. Such similar works are the ones in [58–61]. The corresponding studies investigate the automatic detection of irony based on lexical features as well as the adoption of lexical and pragmatic factors on machine learning effectiveness for identifying sarcastic utterances. Finally, we plan on creating an online service that takes advantage of Spark Streaming, which is an Apache Spark library for manipulating streams of data that provides users with real-time analytics about sentiments of requested topics.

Author Contributions: Andreas Kanavos, Nikolaos Nodarakis, Spyros Sioutas, Athanasios Tsakalidis, Dimitrios Tsolis and Giannis Tzimas conceived of the idea, designed and performed the experiments, analyzed the results, drafted the initial manuscript and revised the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sentiment. Available online: http://www.thefreedictionary.com/sentiment (accessed on 2 March 2017).
2. Wang, X.; Wei, F.; Liu, X.; Zhou, M.; Zhang, M. Topic Sentiment Analysis in Twitter: A Graph-based Hashtag Sentiment Classification Approach. In Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), Glasgow, UK, 24–28 October 2011; pp. 1031–1040.
3. Emoticon. Available online: http://dictionary.reference.com/browse/emoticon (accessed on 2 March 2017).
4. Lin, J.; Dyer, C. *Data-Intensive Text Processing with MapReduce*; Morgan and Claypool Publishers: San Rafael, CA, USA, 2010.
5. van Banerveld, M.; Le-Khac, N.; Kechadi, M.T. Performance Evaluation of a Natural Language Processing Approach Applied in White Collar Crime Investigation. In Proceedings of the Future Data and Security Engineering (FDSE), Ho Chi Minh City, Vietnam, 19–21 November 2014; pp. 29–43.
6. Agarwal, A.; Xie, B.; Vovsha, I.; Rambow, O.; Passonneau, R. Sentiment Analysis of Twitter Data. In *Workshop on Languages in Social Media*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2011; pp. 30–38.
7. Davidov, D.; Tsur, O.; Rappoport, A. Enhanced Sentiment Learning Using Twitter Hashtags and Smileys. In Proceedings of the International Conference on Computational Linguistics, Posters, Beijing, China, 23–27 August 2010; pp. 241–249.
8. Jiang, L.; Yu, M.; Zhou, M.; Liu, X.; Zhao, T. Target-dependent Twitter Sentiment Classification. In Proceedings of the Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Portland, OR, USA, 19–24 June 2011; Volume 1, pp. 151–160.
9. Dean, J.; Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* **2008**, *51*, 107–113.
10. White, T. *Hadoop: The Definitive Guide*, 3rd ed.; O'Reilly Media/Yahoo Press: Sebastopol, CA, USA, 2012.
11. Karau, H.; Konwinski, A.; Wendell, P.; Zaharia, M. *Learning Spark: Lightning-Fast Big Data Analysis*; O'Reilly Media: Sebastopol, CA, USA, 2015.
12. Pang, B.; Lee, L. Opinion Mining and Sentiment Analysis. *Found. Trends Inf. Retr.* **2008**, *2*, 1–135.
13. Hu, M.; Liu, B. Mining and Summarizing Customer Reviews. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, USA, 22–25 August 2004; pp. 168–177.
14. Zhuang, L.; Jing, F.; Zhu, X.Y. Movie Review Mining and Summarization. In Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), Arlington, VA, USA, 5–11 November 2006; pp. 43–50.
15. Zhang, W.; Yu, C.; Meng, W. Opinion Retrieval from Blogs. In Proceedings of the ACM Conference on Conference on Information and Knowledge Management (CIKM), Lisbon, Portugal, 6–10 November 2007; pp. 831–840.
16. Turney, P.D. Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In Proceedings of the Annual Meeting of the Association for Computational Linguistics, Philadephia, PA, USA, 6–12 July 2002; pp. 417–424.
17. Wilson, T.; Wiebe, J.; Hoffmann, P. Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis. In Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP), Vancouver, BC, Canada, 6–8 October 2005; pp. 347–354.
18. Wilson, T.; Wiebe, J.; Hoffmann, P. Recognizing Contextual Polarity: An Exploration of Features for Phrase-level Sentiment Analysis. *Comput. Linguist.* **2009**, *35*, 399–433.
19. Yu, H.; Hatzivassiloglou, V. Towards Answering Opinion Questions: Separating Facts from Opinions and Identifying the Polarity of Opinion Sentences. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, Edinburgh, UK, 11–12 July 2003; pp. 129–136.
20. Lin, C.; He, Y. Joint Sentiment/Topic Model for Sentiment Analysis. In Proceedings of the ACM Conference on Information and Knowledge Management, Hong Kong, China, 2–6 November 2009; pp. 375–384.
21. Mei, Q.; Ling, X.; Wondra, M.; Su, H.; Zhai, C. Topic Sentiment Mixture: Modeling Facets and Opinions in Weblogs. In Proceedings of the International Conference on World Wide Web (WWW), Banff, AB, Canada, 8–12 May 2007; pp. 171–180.

22. Pang, B.; Lee, L.; Vaithyanathan, S. Thumbs up? Sentiment Classification using Machine Learning Techniques. In Proceedings of the ACL Conference on Empirical methods in Natural Language Processing, Philadelphia, PA, USA, 6–7 July 2002; pp. 79–86.
23. Boiy, E.; Moens, M. A Machine Learning Approach to Sentiment Analysis in Multilingual Web Texts. *Inf. Retr.* **2009**, *12*, 526–558.
24. Nasukawa, T.; Yi, J. Sentiment Analysis: Capturing Favorability Using Natural Language Processing. In Proceedings of the International Conference on Knowledge Capture, Sanibel Island, FL, USA, 23–25 October 2003; pp. 70–77.
25. Ding, X.; Liu, B. The Utility of Linguistic Rules in Opinion Mining. In Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, 23–27 July 2007; pp. 811–812.
26. Xavier, U.H.R. Sentiment Analysis of Hollywood Movies on Twitter. In Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), Paris, France, 25–28 August 2013; pp. 1401–1404.
27. Yamamoto, Y.; Kumamoto, T.; Nadamoto, A. Role of Emoticons for Multidimensional Sentiment Analysis of Twitter. In Proceedings of the International Conference on Information Integration and Web-based Applications Services (iiWAS), Hanoi, Vietnam, 4–6 December 2014; pp. 107–115.
28. Waghode Poonam, B.; Kinikar, M. Twitter Sentiment Analysis with Emoticons. *Int. J. Eng. Comput. Sci.* **2015**, *4*, 11315–11321.
29. Chikersal, P.; Poria, S.; Cambria, E. SeNTU: Sentiment Analysis of Tweets by Combining a Rule-based Classifier with Supervised Learning. In Proceedings of the International Workshop on Semantic Evaluation (SemEval), Denver, CO, USA, 4–5 June 2015; pp. 647–651.
30. Barbosa, L.; Feng, J. Robust Sentiment Detection on Twitter from Biased and Noisy Data. In Proceedings of the International Conference on Computational Linguistics: Posters, Beijing, China, 23–27 August 2010; pp. 36–44.
31. Naveed, N.; Gottron, T.; Kunegis, J.; Alhadi, A.C. Bad News Travel Fast: A Content-based Analysis of Interestingness on Twitter. In Proceedings of the 3rd International Web Science Conference (WebSci'11), Koblenz, Germany, 15–17 June 2011; pp. 8:1–8:7.
32. Nakov, P.; Rosenthal, S.; Kozareva, Z.; Stoyanov, V.; Ritter, A.; Wilson, T. SemEval-2013 Task 2: Sentiment Analysis in Twitter. In Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval@NAACL-HLT), Atlanta, GA, USA, 14–15 June 2013; pp. 312–320.
33. Rosenthal, S.; Ritter, A.; Nakov, P.; Stoyanov, V. SemEval-2014 Task 9: Sentiment Analysis in Twitter. In Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval@COLING), Dublin, Ireland, 23–24 August 2014; pp. 73–80.
34. Rosenthal, S.; Nakov, P.; Kiritchenko, S.; Mohammad, S.; Ritter, A.; Stoyanov, V. SemEval-2015 Task 10: Sentiment Analysis in Twitter. In Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval@NAACL-HLT), Denver, CO, USA, 4–5 June 2015; pp. 451–463.
35. Nakov, P.; Ritter, A.; Rosenthal, S.; Sebastiani, F.; Stoyanov, V. SemEval-2016 Task 4: Sentiment Analysis in Twitter. In Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval@NAACL-HLT), San Diego, CA, USA, 16–17 June 2016; pp. 1–18.
36. Lee, C.; Roth, D. Distributed Box-Constrained Quadratic Optimization for Dual Linear SVM. In Proceedings of the 32nd International Conference on Machine Learning (ICML), Lille, France, 6–11 July 2015; pp. 987–996.
37. Zhuang, Y.; Chin, W.; Juan, Y.; Lin, C. Distributed Newton Methods for Regularized Logistic Regression. In Proceedings of the 19th Pacific-Asia Conference, Advances in Knowledge Discovery and Data Mining (PAKDD), Ho Chi Minh City, Vietnam, 19–22 May 2015; pp. 690–703.
38. Sahni, T.; Chandak, C.; Chedeti, N.R.; Singh, M. Efficient Twitter Sentiment Classification using Subjective Distant Supervision. *arXiv* **2017**, arXiv:1701.03051.
39. Kanavos, A.; Perikos, I.; Vikatos, P.; Hatzilygeroudis, I.; Makris, C.; Tsakalidis, A. Conversation Emotional Modeling in Social Networks. In Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Limassol, Cyprus, 10–12 November 2014; pp. 478–484.
40. Kanavos, A.; Perikos, I.; Hatzilygeroudis, I.; Tsakalidis, A. Integrating User's Emotional Behavior for Community Detection in Social Networks. In Proceedings of the International Conference on Web Information Systems and Technologies (WEBIST), Rome, Italy, 8–10 November 2016; pp. 355–362.

41. Baltas, A.; Kanavos, A.; Tsakalidis, A. An Apache Spark Implementation for Sentiment Analysis on Twitter Data. In Proceedings of the International Workshop on Algorithmic Aspects of Cloud Computing (ALGOCLOUD), Aarhus, Denmark, 22–26 August 2016.
42. Nodarakis, N.; Sioutas, S.; Tsakalidis, A.; Tzimas, G. Large Scale Sentiment Analysis on Twitter with Spark. In Proceedings of the EDBT/ICDT Workshops, Bordeaux, France, 15–18 March 2016.
43. Khuc, V.N.; Shivade, C.; Ramnath, R.; Ramanathan, J. Towards Building Large-Scale Distributed Systems for Twitter Sentiment Analysis. In Proceedings of the Annual ACM Symposium on Applied Computing, Gyeongju, Korea, 24–28 March 2012; pp. 459–464.
44. Apache Spark. Available online: http://spark.apache.org/ (accessed on 2 March 2017).
45. MLlib. Available online: http://spark.apache.org/mllib/ (accessed on 2 March 2017).
46. Nodarakis, N.; Pitoura, E.; Sioutas, S.; Tsakalidis, A.; Tsoumakos, D.; Tzimas, G. kdANN+: A Rapid AkNN Classifier for Big Data. *Trans. Large Scale Data Knowl. Cent. Syst.* **2016**, *23*, 139–168.
47. Davidov, D.; Rappoport, A. Efficient Unsupervised Discovery of Word Categories Using Symmetric Patterns and High Frequency Words. In Proceedings of the International Conference on Computational Linguistics, Sydney, Australia, 17–21 July 2006; pp. 297–304.
48. Bloom, B.H. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* **1970**, *13*, 422–426.
49. Using Hadoop for Large Scale Analysis on Twitter: A Technical Report. Available online: http://arxiv.org/abs/1602.01248 (accessed on 2 March 2017).
50. Toutanova, K.; Klein, D.; Manning, C.D.; Singer, Y. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of the HLT-NAACL, Edmonton, AB, Canada, 31 May–1 June 2003; pp. 252–259.
51. Twitter Developer Documentation. Available online: https://dev.Twitter.com/rest/public/search (accessed on 2 March 2017).
52. Go, A.; Bhayani, R.; Huang, L. *Twitter Sentiment Classification Using Distant Supervision*; CS224N Project Report; Stanford University: Stanford, CA, USA, 2009; pp. 1–6.
53. Sentiment140 API. Available online: http://help.sentiment140.com/api (accessed on 2 March 2017).
54. Cheng, Z.; Caverlee, J.; Lee, K. You Are Where You Tweet: A Content-based Approach to Geo-locating Twitter Users. In Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), Washington, DC, USA, 25–28 July 2010; pp. 759–768.
55. Twitter Cikm 2010. Available online: https://archive.org/details/Twitter_cikm_2010 (accessed on 2 March 2017).
56. Twitter Sentiment Analysis Training Corpus (Dataset). Available online: http://thinknook.com/Twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/ (accessed on 2 March 2017).
57. Ternary Classification. Available online: https://www.crowdflower.com/data-for-everyone/ (accessed on 2 March 2017).
58. Barbieri, F.; Saggion, H. Modelling Irony in Twitter: Feature Analysis and Evaluation. In Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC), Reykjavik, Iceland, 26–31 May 2014; pp. 4258–4264.
59. Bosco, C.; Patti, V.; Bolioli, A. Developing Corpora for Sentiment Analysis: The Case of Irony and Senti-TUT. *IEEE Intell. Syst.* **2013**, *28*, 55–63.
60. González-Ibáñez, R.I.; Muresan, S.; Wacholder, N. Identifying Sarcasm in Twitter: A Closer Look. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL), Portland, OR, USA, 19–24 June 2011; pp. 581–586.
61. Reyes, A.; Rosso, P.; Veale, T. A Multidimensional Approach for Detecting Irony in Twitter. *Lang. Resour. Eval.* **2013**, *47*, 239–268.