BIG DATA AND SENTIMENT CLASSIFICATION

Hanan Ather[1],Patrick Boily[1,2,3,4]

Summary
In this paper we use NLP techniques to solve a binary classification problem of determining the sentiment of movie reviews from the IMDB Movie Database and Amazon product reviews. We find that, simpler models such as logistic regression perform better on the test split, however, tree based models are more generalizable on new data which the models were not trained on.

Keywords
Big Data, Spark, Sentiment Classification, TF-IDF, Distributed Computing

[1]Department of Mathematics and Statistics, University of Ottawa, Ottawa
[2]Sprott School of Business, Carleton University, Ottawa
[3]Data Action Lab, Ottawa
[4]Idlewyld Analytics and Consulting Services, Wakefield, Canada
Courriel: hathe098@uottawa.ca

Table of Contents

1. Introduction

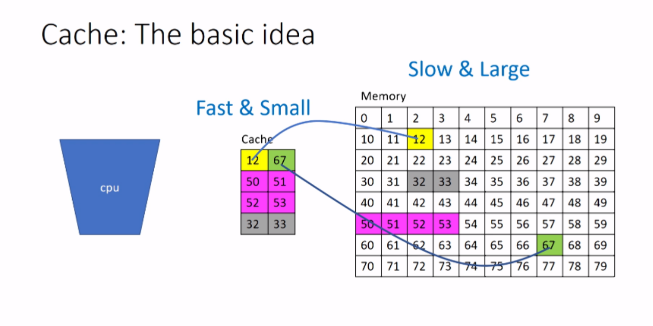
Single machines do not have enough power and resources to perform computations on vast amounts of information. A cluster, or group, of computers, pools the resources of many machines together, giving us the ability to use all the cumulative resources as if they were a single computer. [1]

Now, a group of machines alone is not powerful; you need a framework to coordinate work across them. Spark does just that, managing and coordinating the execution of tasks on data across a cluster of computers. Spark is an open-source distributed cluster computing framework. Distributed computing generally refers to a computational job split across a cluster of nodes. Each node is responsible for a set of operations on a subset of the data. [1] In the end, the partial results are combined together to obtain the final result.

A natural question might be: How do the nodes know which task to run in what order? Are all nodes equal? Which node is the user interacting the code is run? Most distributed computational frameworks are organized into a master-worker hierarchy. The master node is responsible for orchestrating the tasks across the cluster, and the workers are performing the computations. The cluster of machines that Spark will use to execute tasks is managed by a cluster manager like Spark's standalone cluster manager, YARN, or Mesos. [1] We then submit Spark Applications to these cluster managers, which will grant resources to our application so that we can complete our work.

2. Terminology

There are few pieces of hardware which are crucial to understanding big data:



12	67
50	51
52	53
32	33

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79

Figure 1. When the CPU needs access to a particular memory location, it first checks whether it is the cache. If it is, the content can be retrieved very quickly (i.e., a cache hit). Otherwise, if the location is not in the cache, the retrieval will be slow (cache miss) [16].

2.1 Hardware

CPU is said to be the brains of a computer. The CPU functions include performing mathematical calculations and managing other components of the computer. The CPU also can store small amounts of data inside itself in what is referred to as registers. The registers store the data that the CPU is performing a computation on at a given moment. The registers make computations more efficient: the registers avoid sending data unnecessarily back and forth between memory (RAM) and the CPU. [17]

Cache is defined as hardware or software used to temporarily store data in a computing environment [4].

Random Access Memory (RAM) is essentially a machine's short-term memory. Although long term storage such as hard disks and SSD is cheap and durable, its much slower than RAM (or memory). Loading data from magnetic disks can be two hundred times slower, and the solid-state drives are fifteen times slower. [18] For example, processing hours worth of tweets (∼4.3 GB) would take approximately 30 MS if the data is in memory. If the data needs to be loaded from an SSD it will take about 0.5 seconds. Moreover, it would take about 4 seconds if the data needs to be loaded from an older magnetic hard drive [21].

Spark uses a cluster of servers connected by a **network** of servers. Moving data across the network from one machine to another is the most common bottleneck in big data. Network speeds depend on many factors such as internet speed etc. However, on average, it takes about 20x longer to process data when it needs to be downloaded from another machine first [21]). For this reason, distributed systems try to minimize moving data back and forth between nodes of a cluster.

2.2 Storage Latency

Disk Oriented methods for memory storage are becoming unwieldy: they do not scale elegantly to meet the needs of big data [3].

In storage systems, **latency** refers to how long it takes for a single data request to be processed and the correct data found and accessed from the storage media [7].

Memory latency is the time differing between at which the CPU issues a read or write command, and the time the command is completed. This time is very short if the data is in the L1 cache, but very long, relatively speaking if the data is in external memory (SSD). The time it takes to complete each step is called the **step latency.** With Big Data, most latency is due to reading and writing data—not computation. Storage latency varies widely; it mainly depends on the hardware being used. One of the main challenges in big data analytics is minimizing storage costs while maximizing speed [16].

In computer systems, the locality of reference refers to the situation in which the computer tends to access the same set of memory locations repetitively over a short time-period [2]. Access locality refers to the ability of software to make good use of the cache [16]. One can visualize memory as being broken down into pages. Software that uses the same neighboring pages repeatedly (i.e., exhibits strong locality of reference) is said to have good access locality. Modern hardware is specifically designed to optimize and speed up such software.

Spatial and Temporal locality are two types of locality of reference. **Spatial locality** is defined as when storage locations that are in nearby memory locations are accessed or executed in a short time period. **Temporal locality** is characterized as if a particular memory location is accessed at a particular time; then, it is likely that the same location will be reaccessed in a "short" time period.

Spatial and temporal locality describe two different characteristics of how programs access data. However, there are other additional types of locality of references, such as branch locality and equidistant locality, which file storage systems leverage to optimize speed. However, they are beyond the scope of this paper.

The following two examples make the abstract definitions more concrete:

[Insert examples of Spatial and Temporal locality]

2.3 In Memory Processing

During in-memory computation, the data is kept in the random access memory (RAM) instead of disk drives (such as SSD) and is processed in parallel. Keeping data in memory improves the performance by orders of magnitudes. Spark's primary abstraction is RDD and can be cached by the `cache()` or `persist()` command. When we use the `cache()` method in Spark, all the RDDs are stored in memory (RAM). If the data can not fit in memory, it is either stored on disk or recalculated. Therefore, when we need data to analyze, it is already available on the go or can easily be retrieved. This way, the data becomes highly accessible. This reduces the time complexity of running machine learning algorithms on the data by orders of magnitude. [21]

2.4 MapReduce and It's Limitations

MapReduce is a programming model for manipulating large data sets [22]. It is usually composed of three steps: Map, shuffle, Reduce. MapReduce works by first partitioning a large dataset and distributing it across a cluster. The data is analyzed and converted into a key-value pair in the Map step [22]. In the shuffle step, the data is redistributed by the worker nodes. The key-value pairs are shuffled so that all the keys are on the same machine. In the reduce step, the worker nodes process each group of output data, in parallel, and values with the same keys are combined. [22] The limitations of MapReduce created the need for Spark [15] . For operations such as Filter and Joins, one might need to rewrite the jobs, which becomes complex due to the key-value pattern. This pattern is required to be followed in Reducer and Mapper codes. Furthermore, MapReduce is limited for iterative program execution. Some algorithms, such as K-means, require data to be processed iteratively to refine results. In addition to that, MapReduce does not work well with large data on a network. It cannot process a lot of data requiring shuffling operations over the network efficiently. [15]

3. Framework For NLP Project

3.1 Use Cases for Spark

A considerable length of this paper has been dedicated to highlighting the advantages of Spark. It should be made clear that it need not always be used. Spark is meant for big data sets that cannot fit into one computer or need to leverage multiple cores of a machine. If one is working with smaller datasets, R and Python Anaconda environments are almost always the better choices due to the sheer variety of open-source packages available. However, it should be noted that ML libraries such as Scikit-learn are only optimized for using single core processing. They do not fully leverage the power of multi-core machines by default. Other packages allow Scikit-learn algorithms to utilize multiple cores but can run into scale issues if the data becomes too large. However, since Spark is designed to keep data in memory, it is particularly useful for iterative algorithms such as logistic regression or PageRank. Since these algorithms calculate parameters by repeating computations with slightly different parameters, over and over on the same data stored in memory. [21]

3.2 How Spark Works

Spark Applications are composed of a **driver** and a set of **executors**. The driver process executes the main() function. It is deployed on a node in the cluster, and is responsible for three things:

- maintaining information about the Spark Application;
- responding to a user's program or input;
- analyzing, distributing, and scheduling work across the executors [1];

The driver process is absolutely essential—it's the heart of a Spark Application and maintains all relevant information during the lifetime of the application. The **executors** are responsible for carrying out the computational workload that the driver assigned them. This implies that each executor is responsible for only two things:

- executing code assigned to it by the driver [1];
- reporting the state of the computation on that executor back to the driver node. [1];

3.3 TF-IDF

One way to mathematically measure how concentrated into relatively few documents are occurrences of a given word is using TF-IDF algorithm. The TF-IDF algorithm is the product of two terms. The first part is the term frequency (TF) of word t in the document d. Generally, we apply a log_{10} transformation to the count in order to "squash" the raw frequency of the term. The intuition for the log transformation is that a word appearing appearing x number of times is not x times more likely to be relevant to the meaning of the document [11]. Furthermore, since we are applying a logarithmic transformation to sparse vector we must add one to ensure that we do not take log of zero. Thus

$$tf_{t,d} = log_{10}(count(t,d)+1)$$

The second part is inverse document frequency (IDF). IDF is defined by a fraction of the following two terms: N total number of documents, and df_t the number of documents in which term t occurs. Again, a log_{10} transformation is applied to the fraction in order to normalize or "squash" the raw fraction [11]. Thus,

$$w_{t,d} = tf_{t,d} \times idf_t$$

Hence the resulting definition of a TF-IDF weight value $w_{t,d}$ for a word t in document d is

$$w_{t,d} = tf_{t,d} \times idf_t$$

Upon further inspection we can note that the TF-IDF weighting algorithm has the following properties:

- t has the highest weight when t occurs many times within a small number of documents; [12]

- the weight of term t is offset when the t occurs fewer times in a document, or occurs in many documents (the whole corpus); [12]
- lowest when the term t occurs virtually in all documents; [12]

We may view each document as a vector where each component of the vector corresponds to each term in the dictionary or corpus. The idea of creating vector embeddings of words is to represent a word as a point in a multidimensional "semantic space".

4. IMDB dataset

4.1 The Data set

The IMDB dataset was initially published in the paper, Learning Word Vectors for Sentiment Analysis, by Andrew L. Maas et al. The dataset consists of 25,000 positive movie reviews and 25,000 negative reviews. Our training set consisted of 12,500 positive reviews and 12,500 negative movie reviews.

Below is an example of one of the reviews in the training set, which gives an idea of what the data is like:

"*Liked Stanley Iris very much. Acting was very good. Story had a unique and interesting arrangement. The absence of violence and sex was refreshing. Characters were very convincing and felt like you could understand their feelings. Very enjoyable movie.*"

We have no information on what date the review was posted, who posted the review, or any other additional features. This dataset contains the raw text of the review and the review sentiment (positive or negative).

We cleaned and normalized the data by applying various standard functions which, lowercased all words, removed stop-words, punctuation, and symbols, etc.. (refer to IMDB reviews.ipynb notebook for details)

We utilized Spark MLib to apply the TF-IDF weighting scheme, which we discussed earlier in the paper to transform our tokenized reviews into weighted vectors. These vectors were the input to our machine learning models.

This paper employs six commonly used algorithms for classification: Logistic Regression, Decision tree, Random Forest, Gradient-Boosted Trees, Support Vector Machine, and Naive Bayes. Spark MLlib implements a simple distributed version of stochastic gradient descent. Logistic regression algorithms take as input a regularization parameter, along with various parameters associated with stochastic gradient descent, and support all three possible regularizations (none, L1 or L2). Furthermore, Spark MLib allows for distributed training for tree-based methods (Decision Trees, Random Forrest, and Gradient-Boosted Trees) by implementing partitions data by rows, allowing distributed training with millions or even billions of instances.

For more details on the distributed implementation of Spark MLib: https://spark.apache.org/docs/latest/mllib-linear-methods.html

Our general hypothesis before we evaluated any models' performance was: Although simple models such as decision trees and logistic regression are great for their simplicity and interpretation, they are more limited in their power to learn complicated rules and scale to large data sets. Therefore, a one-hundred tree random forest model or gradient-boosted trees would likely perform better. This point of view was based on a case study where logistic regression performance was compared to SVM for sentiment classification [9].

4.2 Classification Metric

The ROC curve (receiver operator characteristic curve) is a plot that displays the performance of a classification model at all classification threshold (Machine Learning Crash Course, Google). The curve has two parameters: true-positive rate (TPR), which is the y-axis of the ROC curve (also referred to as sensitivity), and false-positive rate (FPR), which is the x-axis (also referred to as specificity).

The TPR tells us the proportion of positive reviews that were correctly classified. The FPR tells us the proportion of negative reviews that were incorrectly classified as positive reviews. The goal of a good model is to maximize the TPR and while minimizing the FPR.

The diagonal line represents where the TPR equals the FPR. Any point on the diagonal line represents that the number of correctly identified positive reviews is the same as incorrectly classified negative reviews. To compare different classifiers, it can be convenient to summarize their performance into a single score. The "Area under the ROC curve" (AUC) takes the integral of the entire ROC curve and measures the area under the curve. It is equivalent to the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance, i.e. it is equivalent to the two-sample Wilcoxon rank-sum statistic. [10]. However, it should be noted that there are disadvantages to using AUC, because it does not account for different misclassification costs arising from false-negatives and false-positives [10].

4.3 Results

The training set which the six classifiers were trained on consisted of 25,000 reviews (12,500 positive and 12,500 negative). The test set which the performance of the classifiers was evaluated on also consisted of 25,000 movie reviews.

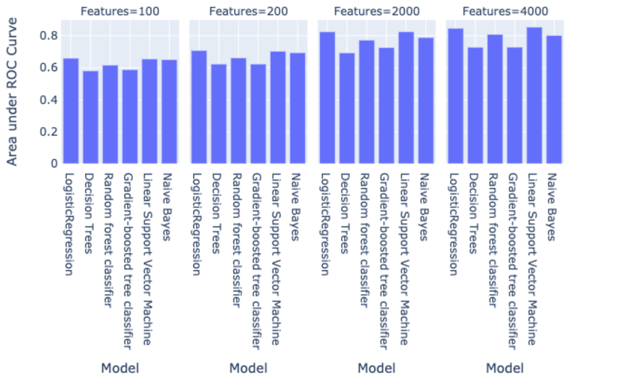


Figure 2. This figure shows the performance of the classifies on 25,000 test set. Logistic regression had the highest AUC for Features= 100, 200, 2000. The AUC for logistic regression was 0.659616, 0.707680, 0.824741. Decision trees had the lowest (worst) AUC of 0.580980, 0.623407, 0.693225, respectively. For features = 4000: linear SVM had the highest AUC of 0.853694, and decision trees had the lowest AUC of 0.728151.

There are two exciting insights that we obtain from the results(see Figure 2). (1) As the number of features increases, all classification models' performance increases on the test set (2) Logistic regression, which is a "simpler" model relative to random forest and gradient-boosted trees perform significantly better. It should be noted at all the models underwent cross-validation to ensure their respective hyper-parameters were optimized.

5. Amazon dataset

After seeing the results, a natural question one might ask is if any of the above models would be good at predicting sentiments of reviews that come from a completely different dataset (or distribution). In the real-word, new data is continually generated, and we might not be guaranteed that the training set resembles the testing set. The next experiment attempted to mimic that situation.

We test the models on a dataset of Amazon product reviews. This data set was significantly different from the IMDB dataset. The main difference is that the data set contained reviews about a product on amazon instead of movies. In other words, the distribution which generated our training data set (IMDB reviews) is different from the distribution which generated the new test set (Amazon Alex Reviews). Therefore, ultimately, we test if sentiment classifiers that perform the "best" in a train/test paradigms are also the most robust classifiers when new data is introduced from a different distribution.

The Amazon Alex dataset consisted of approximately 3000 customer text reviews. Although the dataset contained additional features such as star ratings, date of review, variant, we completely ignored them. The data was obtained from Kaggle Open Datasets. Below is an example of a review in the data set, it gives an idea of what the the data is like:

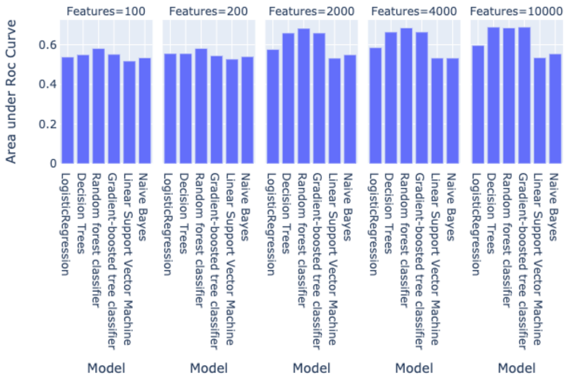


Figure 3. This figure shows the performance of the classifies on 3000 reviews from Amazon. Random forest classier had the highest AUC for Features= 100, 200, 2000, 4000. The AUC for random forest was 0.580840, 0.581025, 0.682131, 0.684670. Linear Support Vector Machine had the lowest (worst) AUC of 0.517603, 0.527168, 0.531929, 0.532542, respectively. For features = 10000: Decision Tree had the highest AUC of 0.688453, and Linear Support Vector Machine had the lowest AUC of 0.534772.

"*We love Alexa! We use her to play music, play radio through iTunes, play podcasts through Anypod, and set reminders. We listen to our flash briefing of news and weather every morning. We rely on our custom lists. We like being able to voice control the volume. We're sure we'll continue to find new uses. Sometimes it's a bit frustrating when Alexa doesn't understand what we're saying.*"

We cleaned and normalized the data by applying various standard functions which, lower casing all words, removing stop-words, punctuation, and symbols, etc.. (refer to 'Alex reviews.ipynb' notebook for details)

Again, we utilized Spark MLib to apply the TF-IDF weighting scheme, discussed earlier in the paper, to transform our tokenized reviews into weighted vectors. These vectors were then inputted to our machine learning models that were created from IMDB reviews training data. The vectors' dimensions were the same as the IMDB dataset (100, 200, 2000, 4000, 10000). The results indicate that Random Forest, Gradient-Boosted Trees, and Decision Trees performed the best on new data(see Figure 3). Logistic regression, linear SVM, and naive Bayes significantly dropped in performance when tested on a data from a new distribution. In essence, models that performed the worst on the IMDB test set performed the best on the Amazon data set and vice versa.

6. Conclusion

From the bias-variance-trade off perspective, logistic regression, linear SVM, naive Bayes classifier had high variance. Although these models performed very well on the test set of the IMDB movie review data set, they do not generalize well on data from a completely different dataset. On the other hand, random forrest, gradient-boosted trees, and decision trees were not the strongest classifier on the IMDB movie review test set, but they generalized much better on the Amazon product review dataset.

7. Future Directions for Research

A natural question to ask is what characteristics of tree-based methods allowed them to perform better on the Amazon data. Random forest and gradient-boosted trees are known to perform well on large NLP datasets, but in our experiment, the decision tree performed just as well. Furthermore, in the next paper, we can use deep learning models to see how well they can generalize. Lastly, in the next paper, we will test if other word embedding algorithms such as word2Vec or doc2Vec give improved results.

References

[1] Chambers Bill, and Matei Zahari. *Spark: the Definitive Guide*. O'Reilly, 2018.

[2] Stallings, William. *Computer Organization and Architecture: Designing for Performance*. Pearson, 2019.

[3] Ousterhout, John, et al. "The Case for RAMClouds." *ACM SIGOPS Operating Systems Review*

[4] Rouse, Margaret. **"What Is Cache (Computing)?"** ,*SearchStorage*, TechTarget, 7 Aug. 2018

[5] Siddhartha, Manu. **Amazon Alexa Reviews** , *Kaggle*, 31 July 2018

[6] Benjamin, and Jenny Kim. **"Data Analytics with Hadoop."**, *O'Reilly Online Learning*, O'Reilly Media, Inc.

[7] **"What Is Latency? And How Is It Different from IOPS?"**, 30 June 2020

[8] **Classification: ROC Curve and AUC Machine Learning Crash Course."** , *Google*

[9] Dwyer, Gareth. **Comparing Support Vector Machines and Decision Trees for Text Classification.**,*Codementor*

[10] Hajian-Tilaki, Karimollah. **Receiver Operating Characteristic (ROC) Curve Analysis for Medical Diagnostic Test Evaluation.**, *Babol*,*Caspian Journal of Internal Medicine*, Babol University of Medical Sciences, 2013

[11] Dan Jurafsky and James H. Martin. **Speech and Language Processing (3rd Ed. Draft)**, *Codementor*

[12] Christopher D. Manning, and Prabhakar Raghavan **Introduction to Information Retrieval**, *Codementor*

[13] Halligan, Steve, et al. **Disadvantages of Using the Area under the Receiver Operating Characteristic Curve to Assess Imaging Tests: a Discussion and Proposal for an Alternative Approach."** , *European Radiology*, Springer Berlin Heidelberg, Apr. 2015

[14] **Linear Methods - RDD-Based API**,*Linear Methods - RDD-Based API - Spark 3.0.0 Documentation*

[15] **MapReduce Tutorial**,*Hadoop*

[16] edX's **Big Data Analytics Using Spark**

[17] Udacity's **Spark Course**, Lecture 2, Hardware module: CPU.

[18] Udacity's **Spark Course**, Lecture 2, Hardware module: Memory.

[19] Udacity's **Spark Course**, Lecture 2, Hardware module: Storage.

[20] Udacity's **Spark Course**, Lecture 2, Hardware module: Network.

[21] Udacity's **Spark Course**, Lecture 2, Hardware module: Big Data Numbers.

[22] Udacity's **Spark Course**, Lecture 2, Hardware module: MapReduce.

[23] Team, DataFlair. **Spark In-Memory Computing - A Beginners Guide."** , 30 Jan. 2019.