A Feasibility Study on Implementing k-NN and Naive Bayes from Scratch using PySpark

Yujia Zheng, Xiyao Wang, KaYat Liu
Master of Data Science and Artificial Intelligence
y326zhen@uwaterloo.ca
ky5liu@uwaterloo.ca
x858wang@uwaterloo.ca

Abstract

Large amounts of data are being generated every day and addressing big data posts challenges in terms of time and the size of the computational infrastructure. Spark is designed to process big data in parallel to achieve fast speed and the generality to handle a variety of tasks. In this research paper, we examined the feasibility of implementing various word embedding methods and algorithms from scratch in PySpark and their performance. Our work is summarized in this workflow. We applied them onto the SMS spam dataset for demonstration and achieved an outstanding accuracy of 98.57% using Naive Bayes model with TF-IDF embedding methods and a short runtime of less than 30 seconds in total on our Naive Bayes models with Bag of Words and TF-IDF embedding methods.

1 Introduction

1.1 Spark

With the great amounts of data that are generated every day, addressing big data requires a large computational infrastructure for successful data processing and analysis (McAfee et al. [1]). Hence, researchers have been looking for a cost-effective solution to handle and process big data, and MapReduce was proposed for such challenges. MapReduce tackles the problem by allowing large datasets to be processed parallelly on a cluster (Dean and Ghemawat [2]). Google has implemented MapReduce in their system since 2004 and processed a total of over 20 petabytes of data per day in 2008 (Dean and Ghemawat [2]).

In 2009, the researchers in the UC Berkeley RAD Lab realized that MapReduce was inefficient for iterative and interactive tasks so they started working on Spark as a research project. Making use of the MapReduce mechanism and extending it to support more operations such as interactive queries, Apache Spark is a cluster computing platform that is designed to process big data efficiently. Spark provides faster speed due to in-memory storage to run computations, and greater generality to cover a large range of tasks in the same engine, such as iterative algorithms, interactive queries and streaming, which requires separate distributed systems previously. It has a number of components, such as Spark Core, Spark SQL and Spark Streaming, to support various tasks (Karau [3]). Our project mainly utilizes Spark Core with the resilient distributed datasets (RDDs) programming abstraction to perform computations in parallel and implement different Natural Language Processing (NLP) algorithms from scratch to process text and carry out predictive modeling, in particular text classification.

Spark is also highly accessible with simple APIs in various programming languages such as Python, Java and Scala ([3]). In our project, we use Python as our programming language with the Python API PySpark to connect with Spark.

1.2 Project Overview

In this research paper, our goal is to implement various word embedding methods and NLP algorithms from scratch using MapReduce in PySpark. We applied the SMS spam dataset for demonstration and achieved an outstanding accuracy of 98.57% on our Naive Bayes models with TF-IDF embedding methods.

In this study, we applied the concept of parallel computing using PySpark with the SMS Spam data set (Almeida et al. [4]). Although the dataset is relatively small compared to the current standard of big data, it can potentially be much bigger and our models would have no issue running on much bigger SMS datasets with the ability to run computations in parallel in Spark and scale up.

In particular, the SMS Spam Collection dataset is extracted from several sources: The Grumbletext Web site, the NUS SMS Corpus, Caroline Tag's Ph.D. Thesis and SMS Spam Corpus v.0.1 Big. The details of the method of extraction are documented on the paper by Almeida et al. [4]. The SMS Spam Collection dataset consists of 4,827 SMS labeled "ham" and 747 mobile spam messages labeled "spam", resulting in a total of 5,574 short messages. The dataset has a total of 81,175 tokens and 7042 distinct tokens. Each SMS has an average of 14.56 tokens.

2 Methods

2.1 Data Manipulation

We noticed that the SMS Spam Collection dataset has an unbalanced number of each label, in which there are 4827 ham texts and 747 spam texts. This may result in a biased model result. Therefore, we sized down the original dataset and created a smaller and balanced dataset which consists of 747 ham texts and 747 spam texts.

2.2 Embedding Methods

We have explored the Bag of Words and TF-IDF text embedding methods to represent the text data, and applied each of them with Naive Bayes and KNN classification algorithms. These methods and algorithms are implemented from scratch in our project and we applied our algorithm on the SMS Spam Collection dataset which can be access here.

2.2.1 Bag of Words

Bag of Words (BoW) is a commonly used text embedding method in natural language processing models. The algorithm is very simple, for a given text, we simply count the number of occurances of each word (Wallach [5]). For example, given the following sentence: Can you can a can like a canner can a can? The BoW embedding is as follow: $\{$can: 5, you: 1, a: 3, like: 1, canner: 1$\}$. Note that in this study we convert all texts into lower case and ignore all punctuations before applying embeddings.

To apply BoW, we first split and tokenize each text in the training set and collect a set of distinct tokens, which is used as the features of BoW. Then for each text, we count the number of times that each token appears. To utilize the parallel computing property of Spark, we need to ensure the representation of each text has the same dimension. Thus, for the features that do not exist in a text, we simply append zeros.

2.2.2 Term Frequency–Inverse Document Frequency

Term Frequency–Inverse Document Frequency (TF-IDF) is another popular text embedding method that measures the relevance of a word in a document in a corpus. It is introduced by Luhn [6] for term frequency (TF) which measures the frequency of a word in a document, and Jones [7] for inverse document frequency (IDF) which measures the importance of a word in a document. For TF, we use raw count in our project that for a word t in a document d, $TF(t, d)$ is the number of occurrence of word t in document d. For IDF, it is defined as $IDF(N, t) = N/log(DF_t)$ where N is the total number of documents in the corpus and DF_t is the number of documents in the corpus that contains the word t. Then the TF-IDF for a word t in a document d for a corpus containing N documents is defined as $TF - IDF(N, t, d) = TF(t, d) * IDF(N, t)$.

To apply the TF-IDF text embedding method in our project, we first tokenize the text data by splitting the sentences into list of words, converting them into lower cases and removing punctuations, then collect the number of occurrences of each word in each document to be our TF. After that, we calculate IDF by computing DF as how many entries we get during our previous process for each word, i.e., the number of sentences each word shows up in, and extract the total number of messages in the text data and apply the formula to get IDF. Lastly, we multiply TF for each word in each sentence with IDF for each word, to get our $TF - IDF$ weight for each word in each sentence. All of the above steps can be calculated in parallel in Spark because the calculations are done independently for each document and for each word and do not require a sequential update.

2.3 Models

2.3.1 K-Nearest Neighbors

The K-Nearest Neighbors (K-NN) is a supervised learning algorithm for classification (Cover and Hart [8]). In the K-NN algorithm, K refers to the number of neighbors to be included in the majority voting process. K is typically a relatively small number and depends on the data. The majority voting process refers to the strategy that a new data point is classified and assigned to the most frequently appearing label among its K nearest neighbors. For example, if K = 5 and we have only 2 classes: *spam* and *ham*. Out of the 5 nearest neighbors of a given datapoint, 3 of them are labeled as *spam* and 2 of them are labeled as *ham*, then this datapoint will be assigned to the label *spam*.

We use the Euclidean distance as the distance metric to find the nearest neighbors. Suppose p and q are 2 datapoints such that $p, q \in \mathcal{R}^m$, where m is the feature dimension. The Euclidean distance is defined as follows:

$$d(p, q) = \sqrt{(p_1, q_1)^2 + (p_2, q_2)^2 + ... + (p_m, q_m)^2} \tag{1}$$

In our study, we compute the Euclidean distance of each test text with every training text using the formula in 1 and store the Euclidean distances and corresponding training text index. To find the K nearest neighbors for each test text, we sort the Euclidean distances in ascending order and find the training text indices corresponding to the top K Euclidean distances. Lastly, we take the majority voting result as the label of each test text.

2.3.2 Naive Bayes

The Naive Bayes algorithm is another supervised machine learning algorithm based on Bayes' Theorem introduced by Thomas Bayes in 1763 (Bayes [9]).

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2}$$

We applied the multinomial Naive Bayes classifier model in our paper with the BoW and TF-IDF embedding methods as our probability in (2) . With our Naive Bayes classifier, we compute the conditional probability of label given the token features, where label takes two values of ham or spam, and $token_1 \ldots token_n$ are the token features (words) in one sms. Then, we compare $P(ham|token_1 \ldots token_n)$ to $P(spam|token_1 \ldots token_n)$, and assign the SMS with the label with a higher conditional probability. Since both posterior probabilities have the same denominator, we can disregard the denominator when comparing. Hence, we obtain (3):

$$\begin{aligned} P(label|token_1 \ldots token_n) &= \frac{P(token_1 \ldots token_n|label) \cdot P(label)}{P(token_1 \ldots token_n)} \\ &\propto P(token_1 \ldots token_n|label) \cdot P(label) \\ &\propto P(token_1|label) \cdot ... \cdot P(token_n|label) \cdot P(label) \end{aligned} \tag{3}$$

The classifier makes sense intuitively as we classify the data with the label that is more likely to happen given the probability tokens in a sentence. However, this method has a big assumption of assuming all features are independent. In other words, we neglect any interactions between words and grammar. However, studies have shown that Naive Bayes performs very well even with the strong assumption (Rish et al. [10]).

Alpha There exists a problem when computing the conditional probabilities at 3. In particular, given 3, if $P(token_i|label) = 0$, $P(token_1 \ldots token_n|label) = 0$ even if $P(token_j|label) \neq 0$, $j\epsilon\{1, \ldots, n \mid j \neq i\}$. Therefore, we introduce a tiny value of 0.0001 for alpha when computing the likelihood to avoid such problems.

3 Results

We successfully achieved our goal of implementing different text embedding methods and NLP algorithms from scratch in Spark with a scalable solution that can handle potentially much bigger data. Below are the results we obtained.

Model	Test Accuracy (%)
K-NN (K=10) with BoW	—
K-NN (K=10) with TF-IDF	—
Naive Bayes with BoW	98.39
Naive Bayes with TF-IDF	98.57

Table 1: K-NN and Naive Bayes Test Accuracy Large Unbalanced Dataset

Model	Test Accuracy (%)
K-NN (K=10) with BoW	77.26
K-NN (K=10) with TF-IDF	50.84
Naive Bayes with BoW	95.32
Naive Bayes with TF-IDF	95.99

Table 2: K-NN and Naive Bayes Test Accuracy Using Small Balanced Dataset

Table 1 and 2 shows the test accuracy of all combinations of embedding methods and models using the large and small datasets. For the models using the large unbalanced dataset, the two Naive Bayes models with BoW and TF-IDF embeddings achieved high test accuracy of 98.39% and 98.57% respectively. However, the two K-NN models with BoW and TF-IDF embeddings both incurred incredibly long runtime, and we failed to collect the test accuracies due to lack of computation resources.

For the models using the small balanced dataset, the Naive Bayes model with TF-IDF embedding attains the highest test accuracy of 95.99%, followed by the Naive Bayes with BoW with 95.32% accuracy. The K-NN model with BoW has a moderate accuracy of 77.26%, however, the K-NN model with TF-IDF has an unexpected low accuracy of 50.84%.

Generally speaking, the models which are trained and tested on the large unbalanced dataset have higher test accuracy. Moreover, our implementation of the Naive Bayes model outperforms the K-NN models in terms of test accuracy and runtime regardless of the embedding methods.

4 Discussion

Our Naive Bayes implementation has achieved outstanding test accuracies. It is entirely based on Spark and thus is scalable when it is needed to handle much bigger data. In future works, we could perform a hyperparameter tuning for the Naive Bayes models to further improve model accuracy.

Our K-NN implementation is entirely based on Spark and is, therefore, scalable when the data becomes bigger. However, the runtime for this algorithm is extremely long because it involves a cross-product every time we calculate the Euclidean distance. In particular, every text token in the test set is paired with every text token in the training set. Therefore, the size of each text message vector is proportional to the vocabulary size of the dataset. Thus, the dimension of the text message vector and the number of rows of the dataset increases substantially, resulting in a long runtime. The inefficiency is due to the fact that K-NN is a lazy model, with the entire dataset being applied every iteration, making it difficult to run in parallel while keeping it efficient. In addition, the inefficiency imposes serious challenges on hyperparameter tuning for K-NN, i.e. finding the optimal choice of K. Therefore, it is better to model K-NN in Python and run sequentially instead of in parallel.

5 Conclusion

We successfully achieved our goal of implementing K-NN and Naive Bayes classifier with Bag of Words and TF-IDF embedding methods from scratch in Spark. Out of all the models, the Naive Bayes model achieved the highest accuracy of 98.57 % in the original dataset, whereas, Naive Bayes with TF-IDF embedding methods achieves the highest accuracy of 95.99 % on the small dataset. The accuracies also resemble the result from the original research paper of the SMS spam dataset by Almeida et al.(Almeida et al. [4]), which further confirms our beliefs that it is feasible to implement some models using Spark from scratch. However, in 4, we discussed the difficulties of implementing some machine learning models. In particular, our K-NN models performed poorly with Spark. Therefore, more considerations should be made before implementing Spark models.

In future studies, we are interested in exploring more machine learning algorithms using Spark. For example, the transformer model is the current state-of-the-art model in the NLP field that is based on the attention mechanism (Vaswani et al. [11]). Nugroho et al. published a paper on a similar topic in 2021 by implementing an attention model, BERT, with spark (Nugroho et al. [12]). Nugroho et al. achieved an average of 62.9% decrease in computation time while decreasing the accuracies by 5.7% (Nugroho et al. [12]). There are still improvements in this topic but we hope this paper could serve as a motivation to popularize Spark-based models.

References

[1] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. "Big data: the management revolution". *Harvard business review*, vol. 90, no. 10 (2012), pp. 60–68 (cit. on p. 1).

[2] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". *Communications of the ACM*, vol. 51, no. 1 (2008), pp. 107–113 (cit. on p. 1).

[3] Holden Karau. *Learning spark : lightning-fast big data analytics*. eng. First edition. Sebastopol, California: O'Reilly, 2015 - 2015 (cit. on p. 1).

[4] Tiago A Almeida, José Marıa G Hidalgo, and Akebo Yamakami. "Contributions to the study of SMS spam filtering: new collection and results". In: *Proceedings of the 11th ACM symposium on Document engineering*. 2011, pp. 259–262 (cit. on pp. 2, 5).

[5] Hanna M Wallach. "Topic modeling: beyond bag-of-words". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 977–984 (cit. on p. 2).

[6] Hans Peter Luhn. "A statistical approach to mechanized encoding and searching of literary information". *IBM Journal of research and development*, vol. 1, no. 4 (1957), pp. 309–317 (cit. on p. 2).

[7] Karen Sparck Jones. "A statistical interpretation of term specificity and its application in retrieval". *Journal of documentation* (1972) (cit. on p. 2).

[8] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". *IEEE transactions on information theory*, vol. 13, no. 1 (1967), pp. 21–27 (cit. on p. 3).

[9] Thomas Bayes. "LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFR S". *Philosophical transactions of the Royal Society of London*, no. 53 (1763), pp. 370–418 (cit. on p. 3).

[10] Irina Rish et al. "An empirical study of the naive Bayes classifier". In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Vol. 3. 22. 2001, pp. 41–46 (cit. on p. 3).

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". *Advances in neural information processing systems*, vol. 30 (2017) (cit. on p. 5).

[12] Kuncahyo Setyo Nugroho, Anantha Yullian Sukmadewa, and Novanto Yudistira. "Large-scale news classification using bert language model: Spark nlp approach". In: *6th International Conference on Sustainable Information Engineering and Technology 2021*. 2021, pp. 240–246 (cit. on p. 5).