Machine Learning with PySpark*

*Note: It is a course project as a requirement for COMP 515

Shukhrat Khuseynov
Graduate School of Science and Engineering
Koç University
Istanbul, Turkey
skhuseynov19@ku.edu.tr

***Abstract*—A distributed programming platform Apache Spark is studied with its machine learning implications, applying the works in its Python version, PySpark. The project consists of two parts: theoretical survey and practical application. The theoretical part is in the form of literature review of the four selected reference papers illustrating the research in the area. The practical part compares and evaluates the distributed implementations of machine learning algorithms via PySpark with the serial implementations via standard Python for three applied business cases of churn estimation, housing price prediction, and sentiment analysis. Both parts are finalized through the four phases, being a requirement for the COMP 515 course.**

Keywords—machine learning, distributed algorithms, Apache Spark, Python, MLlib

I. Introduction

Apache Spark is an efficient data-processing engine designed for big data aiming to improve the scalability and computational speed, especially for data science and artificial intelligence applications. Being based on Resilient Distributed Datasets (RDDs), it is used mainly as a competitive tool for parallel and distributed programming. Although Spark is based on Scala, it has also APIs for Python, R, and Java. The practical implementation is completed in Python, employing PySpark.

Many libraries are available on Spark, such as Spark SQL for structured data and SQL queries, Structured Streaming for stream processing, GraphX for graph analytics, and Spark MLlib for machine learning. The last library, also known as Spark ML in newer versions, is highlighted in the project.

The project describes the application of distributed machine learning algorithms on Apache Spark. The next two sections provide a survey of four papers as a theoretical part and an application of machine learning algorithms on both pure Python and PySpark for the purposes of comparison as a practical part.

II. Literature Review

The first paper [1] presents the MLlib library for distributed machine learning with its main features, strength, benchmarks, performance and scalability measures.

First of all, it is developed in Scala based on C++ linear algebra libraries as a part of Spark ecosystem, being available to many other platforms as an API, including Python. Since the creation of Spark in 2010, it was a demanded library; therefore, it was started and shared in open-source community few years after in 2013.

The library supports the distributed versions of many known machine learning algorithms. Some examples include Naïve Bayes, range of linear models, decision trees in ensembles for regression and classification techniques. There are also algorithms for clustering and dimensionality reduction. MLlib also supports lots of algorithmic optimizations and pipeline API. It also allows Spark integration within its ecosystem, such as SQL, DataFrame abstractions, GraphX, and Spark Streaming. Everything comes with a provided documentation with code dependencies and a good community.

The first evaluation compares the scalability and the speedup of one algorithm, Alternating Least Squares (ALS) in particular, for different platforms. Amazon Reviews dataset was used for scaling on EC2 cluster with 16 nodes. Apache Mahout version 0.9 that uses Hadoop MapReduce was compared with MLlib versions 1.1 and 1.4. As a result, MLlib versions significantly outperform the Mahout/MapReduce, demonstrating great scalability and performance with a better result in a newer version. Furthermore, the second experiment compares MLlib versions 1.0 and 1.1 using linear regression, ridge regression, decision trees, ALS, and KMeans, performed on 16 node EC2 with synthetic datasets. An average of 3x speedup is observed between the versions, with an excellent improvement for logistic regression and the poorest speedup of ridge regression. Therefore, it is a worthwhile library, being in active development.

The second paper [2] compares different platforms for distributed machine learning with detailed evaluations for different types of algorithms. The platforms, such as Apache Spark, PMLS (Parallel ML System), TensorFlow, and MXNet, are described and compared in the paper.

Since big data has more importance nowadays, the usage of large scale distributed machine learning platforms is inevitable. For these purposes a representative dataflow system Spark, a parameter-server system PMLS, more advanced hybrid dataflow systems as Google TensorFlow and MXNet are analyzed with their architectural design in the paper. Each platform has its own advantages and drawbacks.

Evaluations were done on Amazon EC2 cloud computing cluster with Ganglia system monitoring tool, performing

exemplary machine learning techniques as logistic regression and classification of images from MNIST dataset based on feed-forward neural network. For the first task stochastic gradient decent algorithm was utilized and for the second task three models were evaluated: softmax, SNN (single-layer neural network), and MLY (multi-layer neural network).

As a result, the experiments depict that PMLS and MXNet are the fastest and TensorFlow is the slowest for the logistic regression. For the image classification task, where PMLS was omitted, MXNet is slightly faster than TensorFlow for most parts. The Spark is also quite fast for simpler tasks in both cases but becomes slower for more complex techniques. Moreover, experiments for the second task show that Spark has higher CPU and memory utilization and lower network utilization per worker than two other platforms. Although, TensorFlow and MXNet might be slower for some cases, they have good usability and support CPU, GPU, and other devices. Thus, the advantages and the disadvantages of four distributed machine learning platforms were studied.

The third paper [3] proposes optimizations for distributed machine learning applications of Apache Spark. Afterwards, Spark with and without optimizations is compared with MPI framework, evaluating few algorithms.

While frameworks like Hadoop and Spark for distributed programming are more accessible and convenient for the developers, Open MPI (Message Passing Interface) is more flexible with a range of abstractions and primitives for the distributed programming.

Three algorithmic solutions are proposed for dealing with loss minimization problems: the CoCoA algorithm, the distributed minibatch SCD (stochastic coordinate decent), and the distributed minibatch SGD (stochastic gradient decent). They can be used as a part of many other machine learning algorithms and were written from scratch on both Spark and MPI.

After identifying the critical bottlenecks of Spark, two optimizations were suggested to reduce the overhead. Persistent local variables in the memory and meta-RDDs were introduced to the Spark framework coupled with C/C++ for this purpose.

Finally, for the evaluations vanilla Spark, PySpark, Spark combined with C++, PySpark with C++, and MPI were compared and contrasted. Introducing optimizations with C/C++ to Spark/PySpark significantly reduced the time elapsed for running ridge regression trained with CoCoA algorithm on webspam dataset, decreasing the gap with and approaching to MPI's performance. A similar trend was observed with minibatch SCD and SGD, which were slightly slower than CoCoA. In a nutshell, performance difference measured in time between Spark and MPI was decreased from around 20x to 2x thanks to optimizations, which is amazing.

The fourth paper [4] provides a practical case of machine learning task, the sentiment data classification for online reviews using large-scale data, being implemented with distributed machine learning library (MLlib) of Apache Spark, where different algorithms are evaluated and compared.

The paper was motivated due to the fact that there is a lack of research studies that cover distributed machine learning cases using Spark's MLlib library. As a representative customer review data, the Amazon review polarity dataset was deployed, which consists of 2 million reviews of both testing and training sets in total per each class. The reviews were classified either as positive or negative. The negative reviews were lengthier in general, as histogram shows.

Firstly, the data preprocessing by removing null reviews and tokenization of review texts was performed. In addition, the resultant tokens were filtered by removing noise, such as numbers and special characters, and stop-words, such as prepositions and conjunctions. Then, the feature extraction by converting the text tokens into a common measure TF-IDF (term frequency – inverse document frequency) to make them suitable as inputs for the machine learning algorithms, classification models in particular.

Three machine learning classifiers were used: Naïve Bayes, Support Vector Machines (SVM), and Logistic regression. The experiments result in accuracy measures as 85.4%, 86%, and 81.4%, respectively, demonstrating that the SVM classifier had the best performance among three algorithms in terms of accuracy, which is rather inspirational.

III. Practical Implementation

With an inspiration from the fourth reference paper [4], the practical part aims to implement three tasks, using distributed machine learning tools of MLlib on PySpark, compared with their serial versions, implemented by using Sklearn in Python:

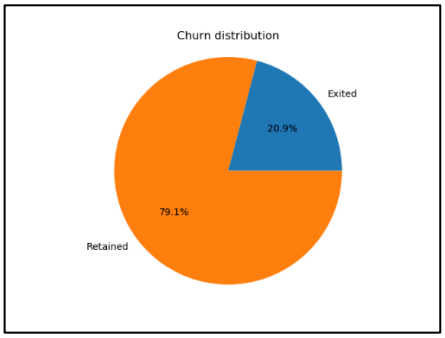
1) churn estimation with a dataset collected from customers of mobile company in the months of 2013 from a real but anonymous source shared on Kaggle (www.kaggle.com/dimitaryanev/mobilechurndataxlsx).

2) housing price prediction with a real dataset of apartment prices in Moscow, being also used in an online course and shared on Kaggle in 2018 (www.kaggle.com/hugoncosta/price-of-flats-in-moscow).

3) sentiment analysis with a real dataset of women's e-commerce clothing reviews, being used in another paper and anonymously shared on Kaggle in 2018 (www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews).

In each case, the datasets are analyzed, providing some meaningful insights and plots, if available. The results of serial and distributed implementations are expected to be slightly different due to randomness and library differences of machine learning models in MLlib (Spark ML) and Sklearn.

The churn estimation involves predicting whether clients will stop doing business with a particular company, representing a binary classification task. The dataset consists of 66469 customer entries with their 66 variables measured. Among the customers 20.9% seem to have churned:



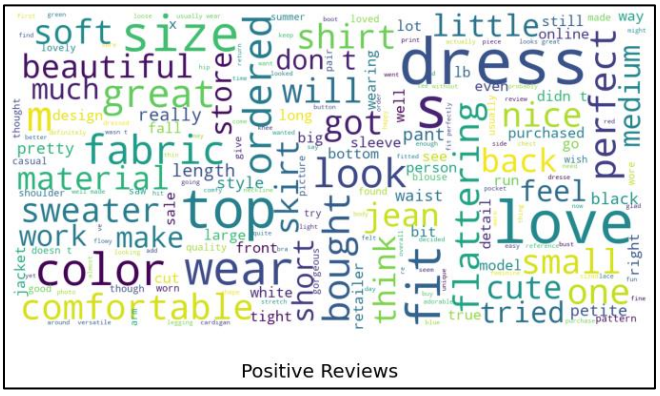
After slight preprocessing, logistic regression and random forest algorithms for the binary classification are performed by training 80% of the dataset and testing the rest. The serial implementation results in accuracies of 86.45% and 86.93%, respectively. Their AUROC or, in other words, the area under the Receiver Operating Characteristic (ROC) curve is 78.18% and 78.43%, accordingly. On the other hand, the distributed version results in accuracies of 86.45% and 87.28% with AUROC of 88.88% and 90.05%, in the same order. Additionally, confusion matrices are reported in the appendix section as a part of the raw model outputs. Both versions have similar accuracies but somehow PySpark outperforms the AUROC measure. In all cases here, the random forest classifier is superior to logistic regression.

The second case describes the prediction of housing prices, given the key details of the apartments, using regression technique for the task. The dataset has 2040 entries with apartment prices and few related variables. After some preprocessing, the machine learning regression task with 80% versus 20% train-test split utilizes linear and random forest regressions. For the serial part, in the former model the Root Mean Square Error (RMSE) error is 29.41 and the correlation coefficient is 82.29%. For the latter, the RMSE score and correlation are 26.18 and 86.42%, accordingly. For the distributed part, RMSE is measured to 27.51 and 27.47, while correlation is 84.06% and 84.10%, respectively. Although, the random forest regressor notably outperforms the linear regression on the pure Python side, on PySpark side both models have very similar performances with random forest being slightly better. Such differences are possibly due to randomness.

The third case involves a sentiment analysis of e-commerce reviews, first implementing the text analysis, then the classification task to find a relationship between the reviews and their recommendations given, classified as positive or negative.

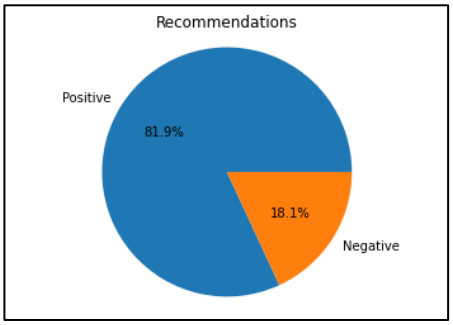
First of all, the dataset is preprocessed by cleaning unneeded columns and null rows. Additionally, the text reviews are lowered to enforce the same letter case and filtered from numbers and special characters, resulting in 22641 review entries with recommendations. These operations are done for both serial and distributed implementation codes, considering the environmental differences of corresponding libraries.

After removing the stop-words, such as prepositions or conjunctions, the word clouds are produced for two classes of recommendations. The tag cloud of positive reviews is depicted below:



Similarly, the tag cloud of negative reviews is demonstrated:

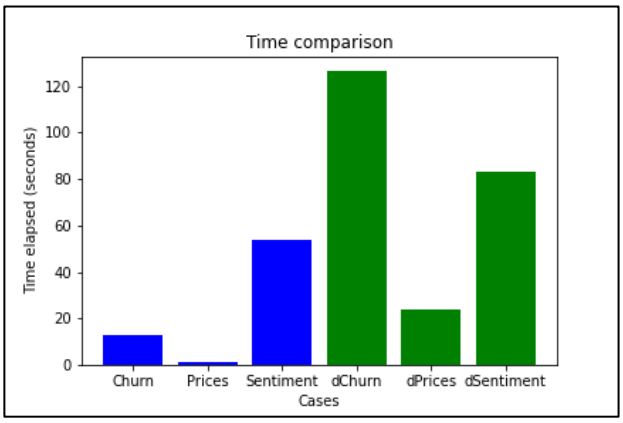
The tag clouds being extracted from preprocessed texts, are combined in accordance with their labels. Although there are many common words in both tag clouds mainly belonging to women's clothes and related words, some words showing clear relation to negative or positive recommendation can be observed. Moreover, among all reviews only 18.1% constitute negative recommendations for the product:



Consequently, the text reviews are tokenized into bag-of-words (BoW) representation, which counts the term frequency (TF), in both PySpark and Python versions. Another common measure TF-IDF, which was mentioned above, is opted out since BoW is observed to be better in preliminary analysis for this case. This way, the BoW representation is used as an input for the modeling.

For the sentiment classification the algorithms of logistic regression and linear support vector machines classifier (SVC) with the 80% training and 20% testing sets are used. As an addition, the Naïve Bayes classifier is also reported in the appendix section for both standard Python and PySpark. In the serial version, two algorithms have accuracies of 88.54% and 88.17% with the AUROC score of 78.29% and 78.45%, respectively. In the distributed version, the accuracy for both is 81.97% and 87.10% with AUROC of 79.62% and 89.83%, accordingly. So, it seems that SVC clearly outperforms logistic regression in PySpark. Although the numbers in serial implementation are quite close to each other, SVC can be concluded as the best in this case, since AUROC measure is more important than accuracy.

Lastly, despite the fact that there are many differences, especially in model specifications and other small details, between serial and distributed implementations, the time for common steps made in both versions mostly is measured for each of three cases to evaluate the general speed difference. The elapsed time is reported in detail in appendix section. Having calculated it for all versions in multiple runs on AWS cluster, the bar plot is given below:



In the figure, the green bars represent the distributed versions. Although the speedup was expected for big data, it seems that the data used are not large enough in size. The smallest dataset is used in housing price prediction case; therefore, it has the lowest speedup (or highest slowdown), as seen in the second bars of above figure. Even though the churn dataset is the largest, the amount of stored data in memory produced throughout the tokenization process of sentiment analysis case is tremendous. This is why the sentiment analysis has the highest speedup (or lowest slowdown).

IV. Conclusion

In a nutshell, the application of machine learning algorithms on Apache Spark, highlighting its Python API, is evaluated both theoretically and practically. The literature review provides different comparisons of Spark with various distributing computing platforms, dataflow systems, and optimized versions in terms of machine learning performance. It also comprises a practical case of distributed sentiment analysis which was a motivation for the practical aspect. The practical implementation applied and compared different machine learning algorithms both serially and in a distributed manner for three business cases, such as churn estimation, prediction of housing prices, and sentiment analysis. The PySpark machine learning methods were widely applied and its timing peculiarities were studied. Larger datasets can potentially be used effectively in PySpark. Such experiments and discoveries inspire for future work in this area.

References

[1] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, “MLlib: Machine Learning in Apache Spark,” *Journal of Machine Learning Research*, vol. 17, no. 34, pp. 1–7, Apr. 2016. Available: https://www.jmlr.org/papers/volume17/15-237/15-237.pdf.

[2] K. Zhang, S. Alqahtani and M. Demirbas, "A Comparison of Distributed Machine Learning Platforms," *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, 2017, pp. 1-9, doi: 10.1109/ICCCN.2017.8038464.

[3] C. Dünner, T. Parnell, K. Atasu, M. Sifalakis and H. Pozidis, "Understanding and optimizing the performance of distributed machine learning applications on apache spark," *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 331-338, doi: 10.1109/BigData.2017.8257942.

[4] S. Al-Saqqa, G. Al-Naymat, and A. Awajan, “A Large-Scale Sentiment Data Classification for Online Reviews Under Apache Spark,” *Procedia Computer Science*, vol. 141, pp. 183–189, 2018. Available: https://www.sciencedirect.com/science/article/pii/S1877050918318167.

APPENDIX

The appendix section includes the raw model outputs for distributed and serial versions of churn estimation, housing price prediction, and sentiment analysis, implemented in practical part.

I. CHURN ESTIMATION (DISTRIBUTED)

Logistic Regression

Confusion matrix:
[[9739 780]
[1031 1818]]
Accuracy: 0.8645272292040694
AUROC: 0.8887937690580443

Random Forest classifier

Confusion matrix:
[[9717 802]
[898 1951]]
Accuracy: 0.8728306403351287
AUROC: 0.9004862784689738

➔ Elapsed time: 126.63735389709473 seconds.

II. CHURN ESTIMATION (SERIAL)

Logistic Regression

Accuracy: 0.8645253497818565
AUROC: 0.7817785968657228
Confusion matrix:
[[9682 770]
[1031 1811]]

Random Forest classifier

Accuracy: 0.8692643297728299
AUROC: 0.7842799953030818
Confusion matrix:
[[9749 703]
[1035 1807]]

➔ Elapsed time: 12.60612940788269 seconds.

III. HOUSING PRICE PREDICTION (DISTRIBUTED)

Linear regression
Correlation: 0.8406081558720544
RMSE: 27.50651961080339

Random Forest regression
Correlation: 0.8410204439870324
RMSE: 27.473998458839905

➔ Elapsed time: 23.571772575378418 seconds.

IV. HOUSING PRICE PREDICTION (SERIAL)

Linear regression

Correlation:
[[1. 0.82291632]
[0.82291632 1.]]
RMSE:
29.41189913898781

Random Forest regression

Correlation:
[[1. 0.86423448]
[0.86423448 1.]]
RMSE:
26.178041773516476

➔ Elapsed time: 0.9875400066375732 seconds.

V. SENTIMENT ANALYSIS (DISTRIBUTED)

Logistic Regression

Confusion matrix:
[[455 359]
[456 3250]]
Accuracy: 0.8196902654867256
AUROC: 0.7961705634398555

Linear Support Vector Machines classifier

Confusion matrix:
[[372 442]
[141 3565]]
Accuracy: 0.8710176991150442
AUROC: 0.8982664409000076

➔ Elapsed time: 82.95256042480469 seconds.

(additional:)
Naive Bayes classifier

Confusion matrix:
[[537 277]
[245 3461]]
Accuracy: 0.8845132743362832
AUROC: 0.532248322993061

VI. SENTIMENT ANALYSIS (SERIAL)

Logistic Regression

Accuracy: 0.8854051667034666
AUROC: 0.7829498776696919
Confusion matrix:
[[540 337]
[182 3470]]

Linear Support Vector Machines classifier

Accuracy: 0.8816515787149481
AUROC: 0.7845213131993091
Confusion matrix:
[[549 328]
[208 3444]]

➔ Elapsed time: 54.05869913101196 seconds.

(additional:)
Naive Bayes classifier

Accuracy: 0.45904173106646057
AUROC: 0.5514984682172247
Confusion matrix:
[[616 261]
[2189 1463]]