**Machine Learning Comparison: Linear SVM (Spark) vs. Non-Linear SVM (SciPy) on the Bank Marketing Dataset**

Jean P. Melendez Villanueva

University of Maryland

Data 660 Advanced Topics in Data Science

Professor Ed Herranz

**Machine Learning Comparison: Linear SVM (Spark) vs. Non-Linear SVM (SciPy) on the Bank Marketing Dataset**

1. **Introduction and Dataset Description**

Support Vector Machines (SVMs) are powerful supervised learning algorithms used for classification and regression tasks. This assignment explores the performance of two SVM variants: a linear SVM implemented using Apache Spark MLlib, and a non-linear SVM implemented using Scikit-learn (SciPy). The objective is to evaluate the accuracy, efficiency, and scalability of both models using the same dataset. By doing so, this project provides a hands-on comparison of model behavior in different computational environments.

The dataset used is the Bank Marketing Campaign dataset from the UCI Machine Learning Repository. It contains 17 attributes, including client demographics, account balance, and details of past marketing interactions. The target variable is binary and indicates whether the client subscribed to a term deposit. This dataset offers a realistic classification challenge that is ideal for comparing linear and non-linear SVMs within both local and distributed machine learning pipelines.

To ensure a fair comparison, both implementations follow similar preprocessing workflows, including data cleaning, feature scaling, and an 80/20 train-test split. In Scikit-learn, categorical variables are one-hot encoded to allow compatibility with the non-linear SVM model. In contrast, the Spark implementation focuses on numeric features to streamline processing within its distributed framework. This structured approach enables a consistent foundation for evaluating the two models and deepens understanding of how different tools handle machine learning pipelines.

1. **Implementation Details**

Apache Spark was used to implement a linear support vector machine (SVM) classifier using the MLlib library. Spark is designed for distributed computing, making it ideal for large-scale data processing tasks that benefit from parallel execution. In this implementation, only numerical features were selected to reduce preprocessing complexity and to align with Spark’s strengths. The selected features were assembled into a single vector and standardized using the StandardScaler to ensure uniform scaling across all inputs.

Once preprocessing was complete, the dataset was split into training and testing sets using an 80/20 random split. The LinearSVC algorithm from Spark MLlib was applied to train the model on the processed training data. After training, predictions were made on both the training and test datasets. Performance was evaluated using custom metrics, including accuracy, sensitivity, specificity, and balanced accuracy.

The model achieved an accuracy of approximately 88.47 percent on the test set. However, it failed to correctly identify any positive cases, resulting in a sensitivity of 0.0000 and a balanced accuracy of 0.5000. This outcome highlights Spark’s limitation in handling class imbalance, as LinearSVC does not support class weighting or kernel transformations. While Spark excels at scalable computation, its default linear SVM is best suited for linearly separable datasets with balanced class distributions.

Spark’s implementation of machine learning models is built on resilient distributed datasets (RDDs) and the DataFrame API, which allow for efficient handling of large volumes of data across multiple nodes. This architecture minimizes memory usage and supports parallel computation without the need for custom distributed programming. For this assignment, while the dataset was relatively small, the Spark pipeline demonstrated how scalable infrastructure can be applied to real-world classification tasks. This reinforces Spark’s role as a powerful tool in big data analytics, even when applied to more modestly sized datasets in academic settings.

1. **Results and Comparison**

Both models achieved similar overall accuracy, but their performance varied significantly in terms of sensitivity and balanced accuracy. The Scikit-learn model, using a non-linear radial basis function (RBF) kernel, achieved an accuracy of 89.06 percent on the test set. In contrast, the Spark Linear SVM achieved an accuracy of 88.47 percent, only slightly lower in terms of correct predictions overall. However, the Spark model failed to identify any positive cases, resulting in a sensitivity of 0.0000 and a balanced accuracy of 0.5000.

This discrepancy highlights the strength of non-linear kernels in capturing complex relationships in real-world datasets. Scikit-learn’s RBF kernel allowed the model to find non-linear boundaries between classes, improving its ability to detect both “yes” and “no” outcomes. Spark’s LinearSVC, limited to linear separation and lacking support for class weighting or kernels, struggled with the imbalanced distribution of the target classes. These results suggest that algorithm choice can have a significant impact when working with unbalanced or non-linearly separable data.

While Scikit-learn is optimized for local execution and offers rich flexibility, it is not designed for distributed computing or large-scale datasets. In contrast, Spark is engineered for scalability, but its built-in algorithms often trade flexibility for performance across clusters. For small to medium-sized classification tasks with complex boundaries, Scikit-learn may produce better results with less configuration. However, Spark remains the preferred choice for large-scale data processing where distributed model training is essential.

1. **Parallelization Discussion**

Parallelization plays a significant role in how machine learning models scale across large datasets and compute environments. According to the Wikipedia article on embarrassingly parallel problems, these are tasks that can be easily divided into independent units with no need for inter-process communication (Wikipedia, 2023). In the context of this project, the linear SVM implementation in Apache Spark fits this definition because the training process can be distributed across multiple nodes without requiring data dependencies between them. Spark’s architecture, built on resilient distributed datasets (RDDs) and task scheduling, makes it well-suited for such embarrassingly parallel operations.

In contrast, non-linear SVMs, like the one implemented in Scikit-learn using an RBF kernel, are not embarrassingly parallel. These models rely on kernel functions that compute the similarity between every pair of data points, resulting in a highly interdependent structure. This leads to increased computational complexity and memory usage, making it more difficult to parallelize the training process without sophisticated coordination. Scikit-learn operates in a single-node environment, and while it can leverage multi-core processors, it does not provide native support for distributed parallelism.

Spark’s design simplifies parallelization by abstracting away the complexity of distributed computing. Developers do not need to manually implement parallel logic; instead, Spark handles data partitioning, task scheduling, and memory management under the hood. This allows models like LinearSVC to be trained efficiently even on very large datasets with minimal code changes. While Spark's linear SVM is limited in flexibility, its ability to distribute workloads across clusters makes it highly effective for scalable machine learning applications.

1. **Conclusion and Insights**

This project demonstrated the practical differences between linear and non-linear support vector machines when applied to the same real-world dataset. Scikit-learn’s non-linear SVM outperformed Spark’s linear model in terms of balanced accuracy and sensitivity, particularly in detecting the minority class. While both models had similar overall accuracy, their behavior differed due to the underlying algorithmic assumptions and feature handling. These results confirm that model choice should be driven by data complexity, class distribution, and computational context.

Spark proved to be highly efficient in terms of scalability and parallelization, but it lacked flexibility in its default linear SVM implementation. Without kernel options or class balancing mechanisms, it struggled with non-linearly separable data and imbalanced target labels. In contrast, Scikit-learn allowed for more customized preprocessing and kernel selection, making it ideal for smaller datasets where nuanced model behavior is required. This distinction highlights the trade-off between scalability and adaptability in machine learning platforms.

In practice, the decision between Spark and Scikit-learn should consider both the nature of the dataset and the infrastructure available. Spark is optimal for large-scale, distributed data pipelines where speed and scalability are essential, while Scikit-learn excels in rapid prototyping and research environments. By working with both tools, this project provided valuable insight into the strengths and limitations of each, as well as a deeper understanding of model behavior under different computational frameworks. Ultimately, applying the right model in the right context is key to achieving effective machine learning outcomes.

**References**

Moro, S., Laureano, R., & Cortez, P. (2014). *Bank marketing data set*. UCI Machine Learning Repository. <https://archive.ics.uci.edu/ml/datasets/Bank+Marketing>

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830. http://www.jmlr.org/papers/v12/pedregosa11a.html

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). *Spark: Cluster computing with working sets*. In Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. <https://spark.apache.org/>

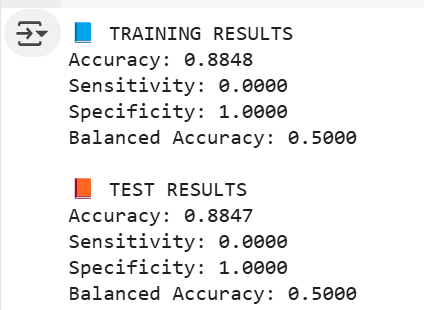
Wikipedia contributors. (2023, March 8). *Embarrassingly parallel*. Wikipedia. <https://en.wikipedia.org/wiki/Embarrassingly_parallel>

**Appendix**

**Appendix A –** Refer to the accompanying Jupyter Notebook file for code implementation

**Appendix B –** Screenshots



(Figure 1) Scikit-learn Accuracy

(Figure 2) Spark Test Accuracy