Implementation of Classification Algorithms Using Apache Spark

Anagha Sarmalkar (801077504) **Shweta Patil (801074059)**

Project Overview :

Our goal was to implement classification algorithms from scratch using Apache Spark, measuring model performance and comparing it with the model executed using Spark's built-in machine learning library MLlib. We have used an airline dataset for predicting whether flights will be delayed or not based on feature parameters.

Motivation:

The motivation for this project was our curiosity behind the workings of machine learning algorithms. We've been accustomed to using algorithms from libraries which posed like black boxes, thus gaining very little understanding of what actually goes under the hood. We designed our own data structures for implementing the algorithms, something we don't have the liberty of while using algorithms from libraries. The boom in Big Data from recent years has made it imperative to leverage Apache Spark's powerful distributed computing framework. Although implementing machine learning algorithms using distributed computing was something we never attempted, it definitely proved to be a richly rewarding experience despite being a challenge. This project has definitely helped us gain more clarity about working with large amounts of data using the powerful tools that are out there.

In order to understand how well our algorithms perform, we decided to test their evaluation metrics using those of those of algorithms from Spark's MLlib library.

Dataset:

In order to make it challenging for our algorithm, we decided to work on the dataset **'2015 Flight Delays and Cancellations'**[1]. This dataset is collected from the U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics which contains information of flights in the year 2015, with respect to airline carrier, original airport, destination airport, distance traveled by the flight, time spent, departure delay time, arrival delay time, etc.

This dataset is significantly large (more than 1 million rows, and 31 features), which makes it ideal for understanding the might of PySpark's distributed computing over Big Data. The rich feature set provided a good exercise in creating pipelines of features and tie the stages together.

Algorithms and Techniques implemented:

Apache Spark is an open source cluster computing framework which has been built around speed, ease of use and streaming analytics. Python is a high level programming language which provides a wide range of libraries majorly used for Machine Learning and real time data analytics. This makes Pyspark, which is a Python API for Spark, harness the simplicity of Python and the high computing power of Apache Spark to process Big Data.

We implemented Logistic Regression with Gradient Descent using PySpark. We used Pipelines to tie the stages of categorical and numerical feature encoding transformations. A pipeline chains multiple Transformations and Estimators together to specify an ML workflow.

We unfortunately could not implement another algorithm due to time constraints.

Implementation

We have followed this approach:

- Exploratory Data Analysis and Feature Engineering

  We prototyped our algorithm on a small dataset first and gradually upscaled the number of records, till we ran out of time to submit the project. Presently, we have tested our algorithm on 100,000 rows which were sampled randomly from the main dataset. We used the following features for our project:

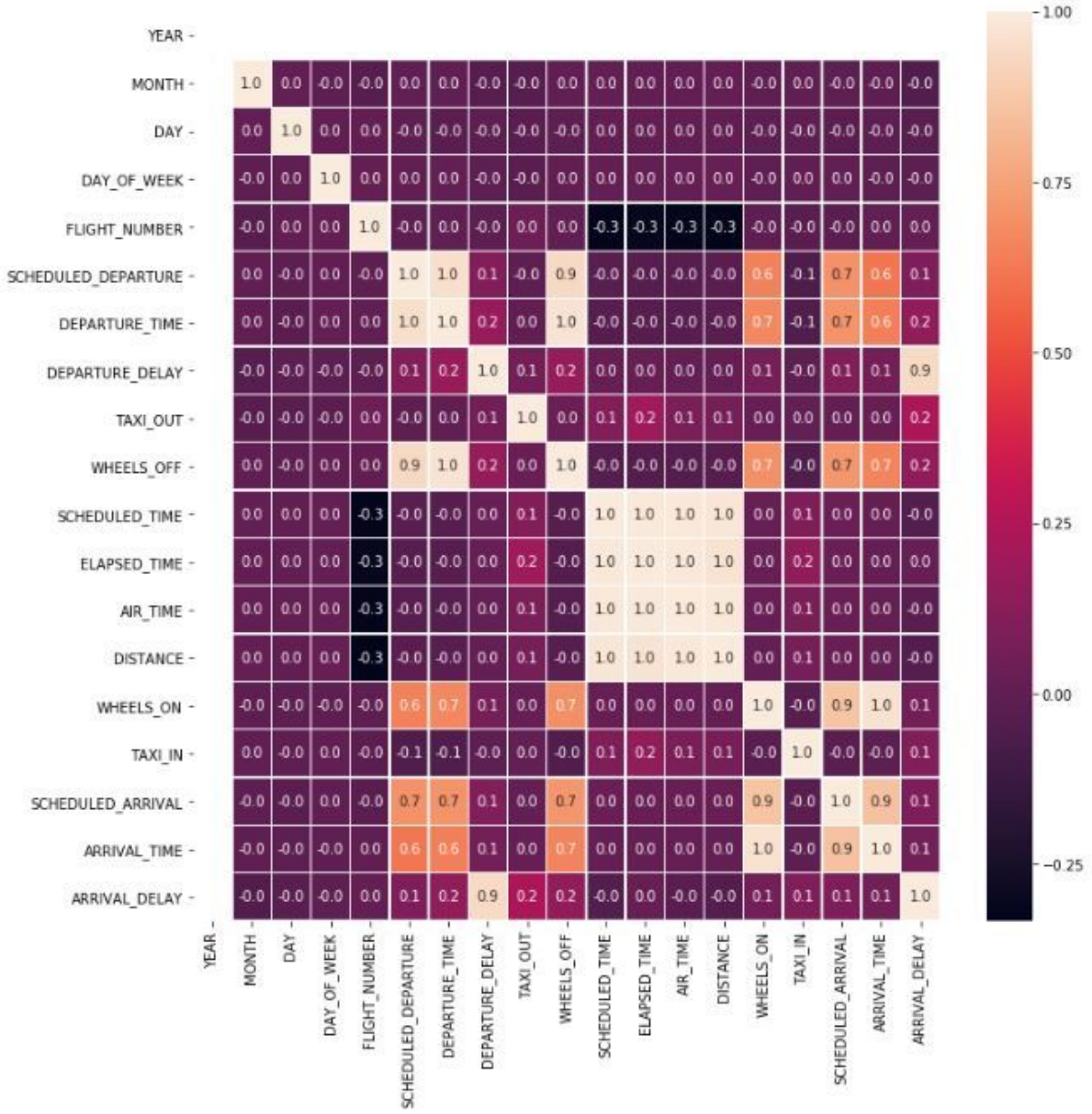
  Input Features:

  MONTH, DAY_OF_WEEK, AIRLINE, ORIGIN_AIRPORT, DESTINATION_AIRPORT, SCHEDULED_DEPARTURE, DEPARTURE_TIME, DEPARTURE_DELAY, TAXI_OUT, WHEELS_OFF, SCHEDULED_TIME, ELAPSED_TIME, AIR_TIME, DISTANCE, WHEELS_ON, TAXI_IN, SCHEDULED_ARRIVAL, and ARRIVAL_TIME

  Output Label:

  ARRIVAL_DELAY

  We have selected the features for positive correlation values.



- Data preprocessing:
  - Cleaning the data for null values and reduced the features.
  - mputed labels for ‘ARRIVAL_DELAY’ delay where values greater than 0 are labelled as “YES” signifying delay and values less than or equal to “NO” signifying no delay.
  - The categorical columns are then encoded using One Hot Encoder and added to stages. The numerical columns are encoded and assembled using VectorAssembler.

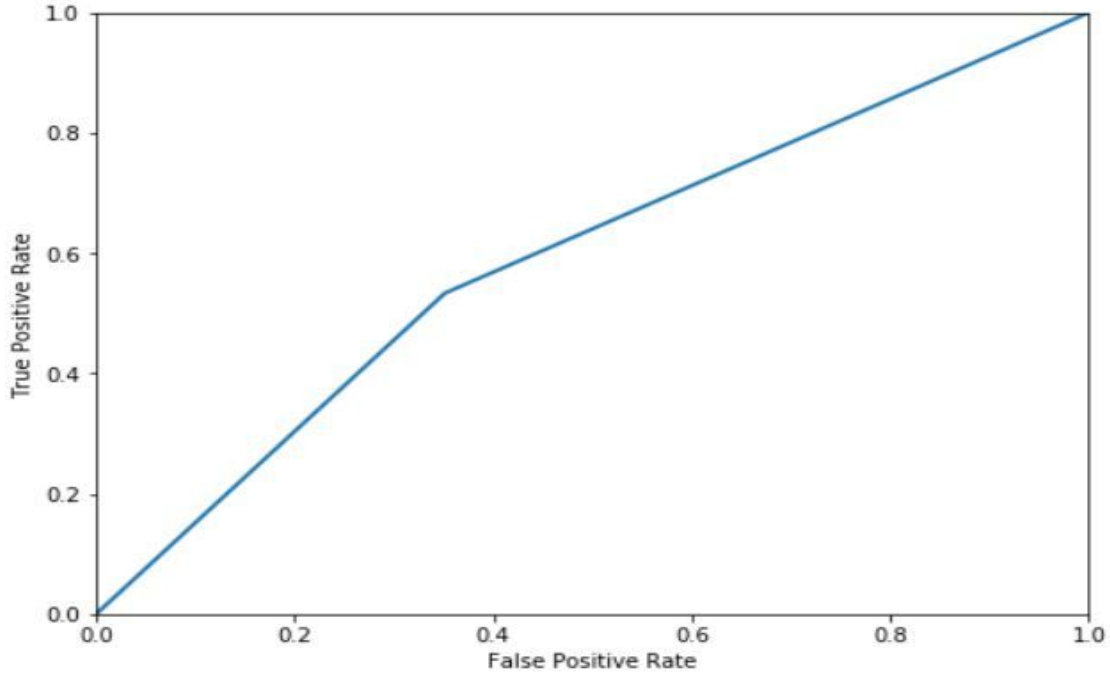
    - These stages added along with the assembler are then inputted to the Pipeline which chains multiple transformations and estimations together.
    - The data is now in the form of a PySpark dataframe with features combined in a vector in features column and labels in a label column. We have added an index column to ensure that the features and the labels are tied to the same index. This helped us during evaluating metrics.
    - Training and Test sets are created by splitting this dataframe randomly in the ratio 7:3
- Implementing Logistic Regression:
    - In order to aid data flow, we persisted all the training and test rdds and unpersisted them at the end of the code.
    - We implemented Logistic Regression with Gradient Descent. The weights and the bias was initially set to 0. Here a pair of input vector and output label (Training) was passed to the functions to calculate the loss function (weight and bias component). These components are combined together and added to be deducted from the initialized weights and bias to update them. This goes on for n iterations.
    - The model is then tested for the test set and also for the Logistic Regression in MLlib for n iterations.

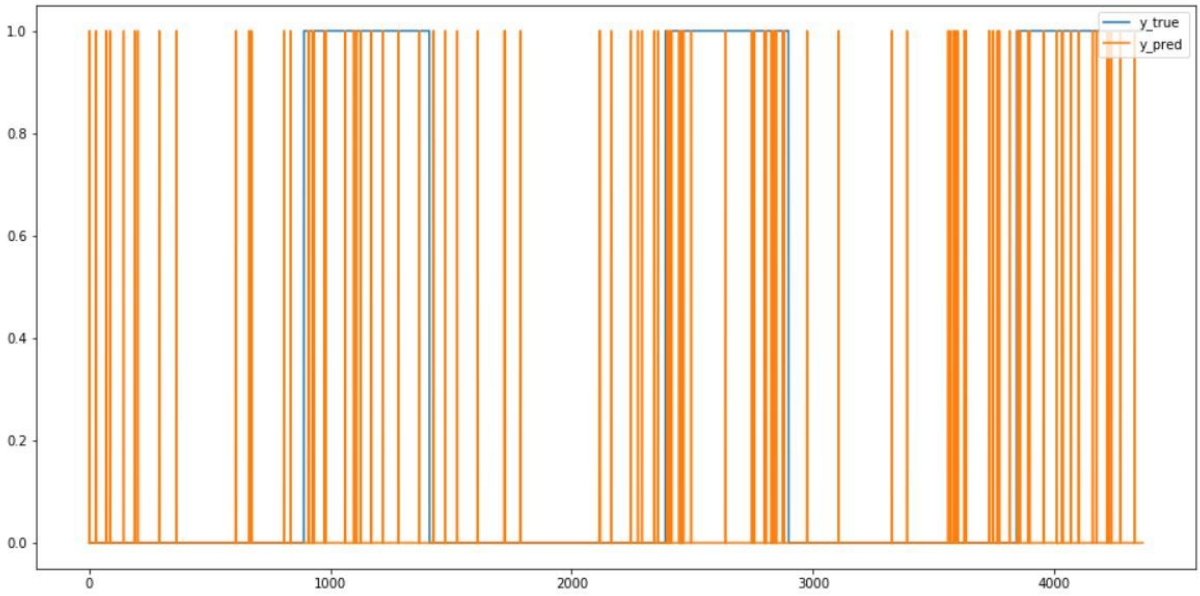
Results :

Logistic Regression (For 15000 records and 500 iterations)

Custom Logistic Regression model statistics	**MLib Logistic Regression model statistics**
Accuracy : 0.6459	Accuracy : 0.8533
Precision : 0.0366	Precision : 0.8593
Recall : 0.5333	Recall : 0.9185
F-measure : 0.0675	F-measure : 0.8879

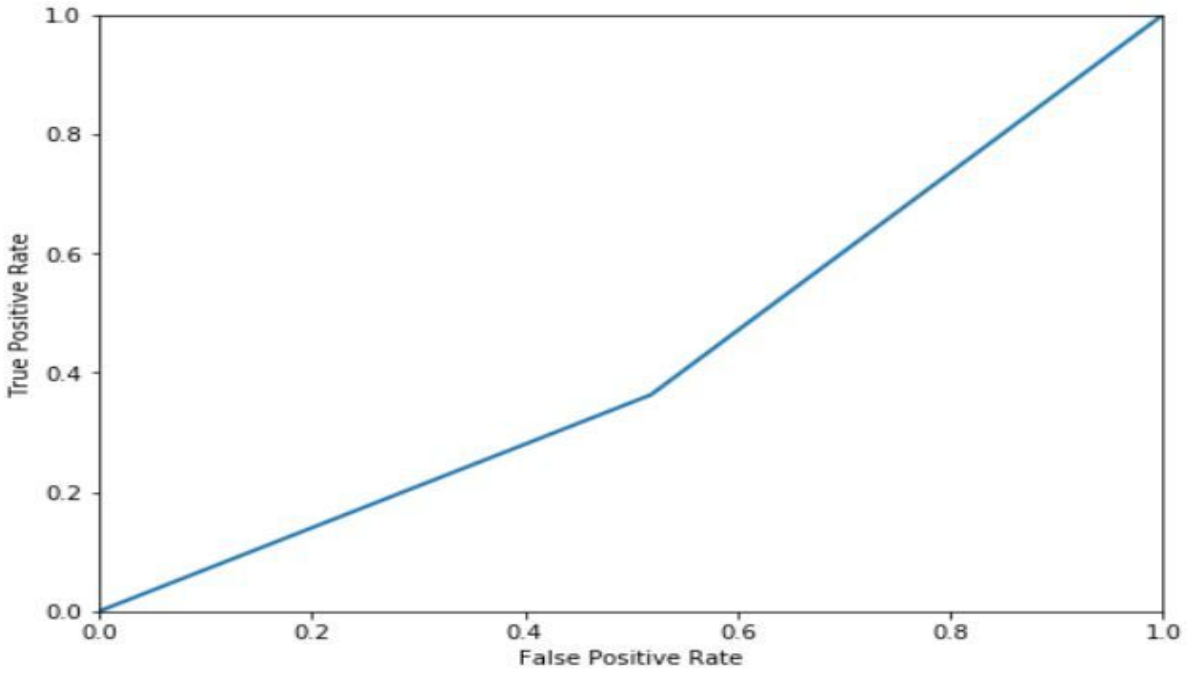
We have plotted graphs of results of custom-built Logistic Regression model as follows -

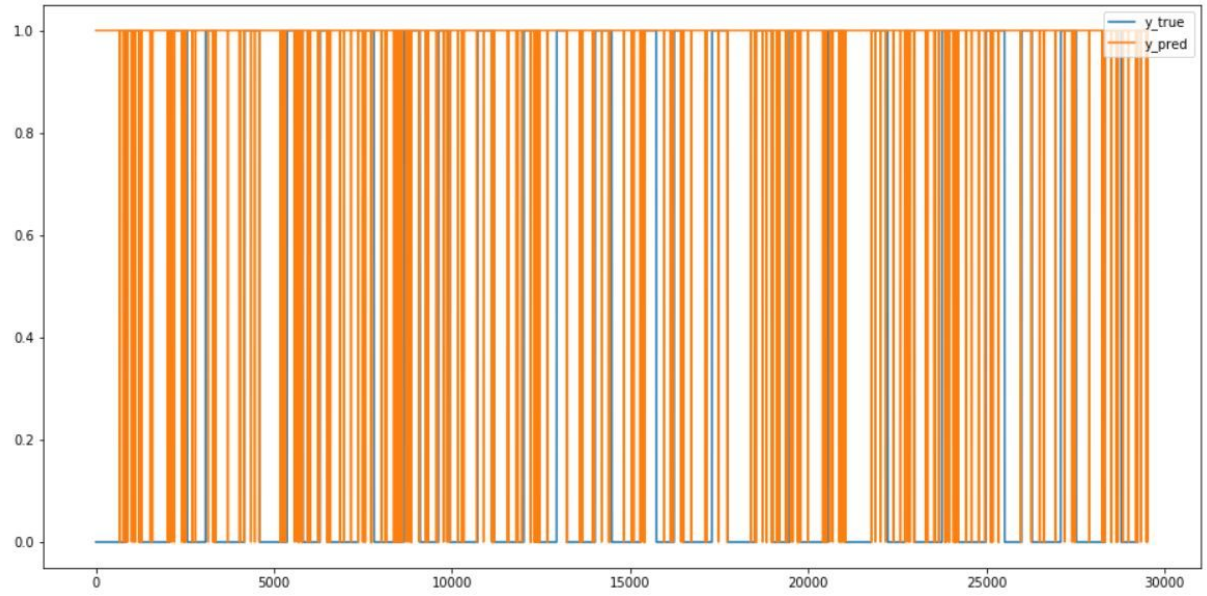




Logistic Regression (For 100000 records and 100 iterations)

Custom Logistic Regression model statistics	**MLib Logistic Regression model statistics**	**MLib Random Forest model statistics**
Accuracy : 0.3639	Accuracy : 0.9903	Accuracy : 0.8148
Precision : 0.9870	Precision : 0.8593	Precision : 0.7862
Recall : 0.3628	Recall : 0.9951	Recall : 0.9701
F-measure : 0.5306	F-measure : 0.9924	F-measure : 0.8685





Accomplishment of aspects in will do, likely will do, and would ideally like to do items:

We accomplished implementing Logistic regression from scratch. We however did not have enough time to implement another algorithm. We instead implemented Random Forest using MLlib.

We wish we could work more on our algorithm to get the metrics closer to that of the library. They are comparable however.

We have not implemented any more classification algorithms.

Division of responsibility

Team Member Name	Task
Anagha Sarmalkar (801077504)	Data Preprocessing, Feature Engineering,implementing Logistic Regression, and Naive
Shweta Patil ((801074059)	Model Evaluations, comparison with algorithms in MLlib,Webpage

References

[1] 2015 Flight Delays and Cancellations https://www.kaggle.com/usdot/flight-delays
[2] https://docs.databricks.com/
[3] https://spark.apache.org/docs/latest/api/python/index.html