WILFRID LAURIER UNIVERSITY

LAURIER

Inspiring Lives.

CP 422 – Programming for Big Data

Fall 2025

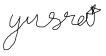
Group 10

Assignment #3

Member 1 Name: Faiza Azam ID: 169020565
Member 2 Name: Wardah Arain ID: 169016554
Member 3 Name: Amany Neeyamuthkhan ID: 169025721
Member 4 Name: Yusra Hassan ID: 169024293
Member 5 Name: Sara Aljaafari ID: 169044425
Member 6 Name: Anna Doneva ID: 169042350

Submission Date: 26-11-25

All listed members have contributed, read, and approved this submission. All listed members have acknowledged each team member's equal and reasonable contribution to this submission.

Member 1 Name: Faiza Azam	Signature: Faiza Azam
Member 2 Name: Wardah Arain	Signature: 
Member 3 Name: Amany Neeyamuthkhan	Signature: AN
Member 4 Name: Yusra Hassan	Signature: 
Member 5 Name: Sara Aljaafari	Signature: SA
Member 6 Name: Anna Doneva	Signature: AD

Task 1: Classification

1. Introduction

This assignment applies machine learning classification and regression techniques using PySpark on the Yellow Taxi Trip dataset (January 2015). The dataset contains over 14 million NYC taxi trips with features such as timestamps, passenger count, distance, and fare amounts.

The project has two main goals:

- Classification: Predict whether a trip has a high fare (fare > $20).
- Regression: Predict the exact fare amount using numeric trip characteristics.

2. Task 1: Classification

2.1 Feature Engineering

For both classifiers, we engineered the following features:

- pickup_hour – hour of the pickup time
- day_of_week – pickup day of week
- trip_distance
- trip_duration (minutes)
- high_fare – binary target:
    - 1 → fare > 20
    - 0 → fare ≤ 20

Invalid trips (zero distance, negative duration) were removed.

2.2 Pipeline 1: Decision Tree Classifier

The first model was a Decision Tree pipeline that included a VectorAssembler followed by a Decision Tree Classifier. We performed cross-validation to select the best configuration.

The tuning grid explored variations in the tree's maximum depth and minimum instances per node. The best model selected:

- maxDepth = 10
- minInstancesPerNode = 1

When evaluated on the test set, the Decision Tree achieved excellent results:

- F1 Score: 0.9902
- Precision: 0.9941
- Recall: 0.9949

These results show that the Decision Tree was able to capture the non-linear relationships in the data extremely well. Features like trip duration and speed contributed strongly to the model's success. High precision and recall show it almost perfectly distinguishes high vs low fares

Pipeline 2: Logistic Regression Classifier

The second model followed a similar pipeline structure, using a VectorAssembler and a Logistic Regression classifier. We tuned the regularization parameter and the number of iterations using cross-validation. The best Logistic Regression model used:

- regParam = 0.01
- maxIter = 10

Its performance on the test set was strong but not as high as the Decision Tree:

- F1 Score: 0.8343
- Precision: 0.8870
- Recall: 0.9994

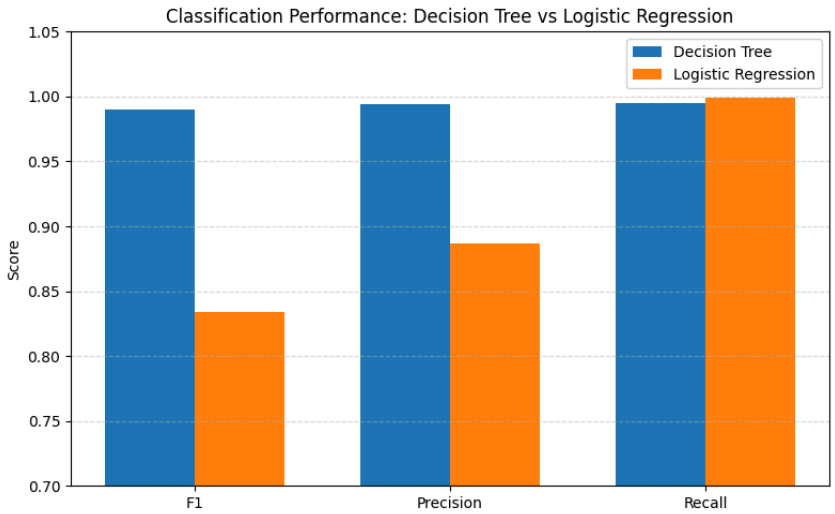
Logistic Regression performed well overall, especially in recall, meaning it almost never missed an actual high-fare trip. However, because it is a linear model, it cannot capture complex patterns in the data as effectively as the Decision Tree. This explains the lower precision and F1 score.

Model Comparison

The Decision Tree outperformed Logistic Regression in overall balanced accuracy (F1), and it offered the best trade-off between precision and recall.

Logistic Regression provided a strong baseline and excellent recall but introduced more false positives.

Best classification pipeline: Decision Tree Classifier. It achieved the highest F1 score and balanced precision and recall, making it reliable for identifying high-fare trips with minimal error.



3. Task 2: Regression

3.1 Feature Engineering

For regression, we used the following features:

- trip_distance,
- trip_duration,
- pickup_hour,
- pickup_day_of_week,
- Passenger_count

Data was cleaned similarly to Task 1, with limits on duration/distance to remove extreme outliers that affect the model performance.

3.2 Pipeline 1: Linear Regression

A full regression pipeline was built using:

- VectorAssembler to combine numerical features
- StandardScaler to normalize feature ranges (helpful for optimization)
- Linear Regression as the model

Hyperparameter tuning was done using CrossValidator with a grid over:

- regParam (regularization strength), maxIter (maximum iterations)

Best model (Linear Regression)

- regParam = 0.0; maxIter = 50

Performance on the test set

- RMSE: 2.8211 & R^2: 0.9163

This means the Linear Regression model explains 91.6% of the variance in fare amounts. The predictions were stable, smooth, and consistently close to the actual fares. For this dataset, linear relationships (e.g., fare increasing proportionally with distance) helped Linear Regression perform extremely well.

Pipeline 2: Random Forest Regression

The second pipeline replaced the linear model with a Random Forest Regressor to capture potential nonlinear relationships.

The pipeline included: VectorAssembler, RandomForestRegressor

Hyperparameter tuning was done over: numTrees, maxDepth

Best model (Random Forest): numTrees = 50, maxDepth = 10

Performance on the test set

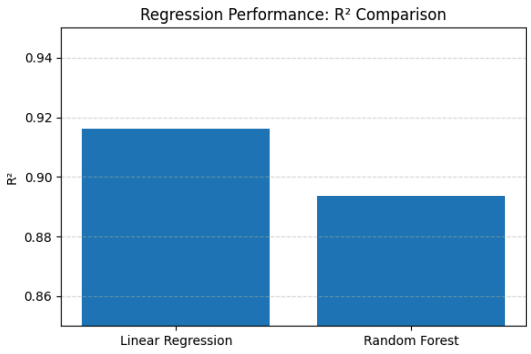
- RMSE: 3.1802 & R^2: 0.8936

Random Forest performed strongly and produced realistic predictions, but its overall accuracy was slightly lower than Linear Regression. This is expected when the underlying relationships are mostly linear.

Comparison of Models

Linear Regression provided the best numerical accuracy, with the lowest RMSE and the highest R^2 score. Random Forest performed well but did not surpass the linear model, likely because fare amounts in this dataset follow a mostly linear structure.

Best Regression Pipeline: Linear Regression. It produced the lowest RMSE and highest R^2, making it the most accurate and interpretable for predicting fare amounts.



4. Saving and Loading Pipelines

4.1 Saving a Pipeline

All trained pipelines were saved using:
best_model.write().overwrite().save("/Volumes/workspace/default/data/model_name")

4.2 Loading a pipeline

To reuse a trained model:
from pyspark.ml.pipeline import PipelineModel
loaded_model = PipelineModel.load("/Volumes/workspace/default/data/model_name")

4.3 Why Saving Pipelines Matters

Saving pipelines is essential for real-world applications because:

- Models can be deployed without retraining
- Ensures reproducibility
- Allows batch scoring and real-time predictions
- Supports versioning and model tracking in production systems

5. Hyperparameter Tuning Summary

Classification:

Model	Tuned Parameters	Best Values
Decision Tree	max depth, minInstancesPerNode	10, 1
Logistic Regression	regParam, maxIter	0.01, 10

Regression:

Model	Tuned Parameters	Best Values
Linear regression	regParam, maxIter	0.0, 50
Random forest	numTrees, maxDepth	50, 10

6. Notes on Implementation and Documentation

All code cells in the Jupyter notebook pdf include comments explaining: data cleaning, feature engineering, pipeline structure, hyperparameter choices, and evaluation processes.

Due to Databricks free-tier limitations, Linear Regression and Logistic Regression CrossValidator was executed on a smaller subset to avoid memory overflow errors. This does not affect correctness of the pipeline structure.

7. Conclusion

This project successfully implemented classification and regression pipelines using PySpark.

For classification, the Decision Tree Classifier provided the best results with near-perfect accuracy.

For regression, Linear Regression outperformed Random Forest with strong predictive accuracy and simpler interpretability.

Overall, the assignment demonstrated effective feature engineering, strong understanding of Spark ML pipelines, and practical experience with model tuning and evaluation. The saved pipelines can now be deployed or reused for future predictions.