Big data - Project 4

Juan Gonzalo Quiroz Cadavid[1], Priit Peterson[1], Shivam Maheshwari[1], and Venkata Narayana Bommanaboina[1]

[1]University of Tartu
juangonzalo@ut.ee, shivammahe21@gmail.com, bvnarayana515739@gmail.com, priit.petersonest@gmail.com

May 11, 2025

Data Ingestion and Preparation (2 pts) 1

We define the data structure Using StructType (See code 1), then 2009 data was used for training and 2010 for testing (See code 2).

Listing 1: Code

```
schema = StructType([
    StructField("FL_DATE", DateType(), True),
    StructField("OP_CARRIER", StringType(), True),
    StructField("OP_CARRIER_FL_NUM", IntegerType(), True),
    ...
    ...
    ...
    StructField("SECURITY_DELAY", DoubleType(), True),
    StructField("LATE_AIRCRAFT_DELAY", DoubleType(), True),
])
```

Listing 2: Code

```

TRAIN_PATH = "input/2009.csv"
TEST_PATH  = "input/2010.csv"
flights_2009 = spark.read.format("csv") \
    .option("header", "true") \
    .option("ignoreLeadingWhiteSpace", "true") \
    .option("ignoreTrailingWhiteSpace", "true") \
    .schema(schema) \
    .load(TRAIN_PATH)

test_df = spark.read.format("csv") \
    .option("header", "true") \
    .option("ignoreLeadingWhiteSpace", "true") \
    .option("ignoreTrailingWhiteSpace", "true") \
    .schema(schema) \
    .load(TEST_PATH)
```

Cleaning and Preprocessing (2 pts) 2

Our first step was renaming the columns for later usage (See code 3). Then, we enhance the timestamp attributes by creating day of the week and month, which will be use later (See code 4).

We inspect the data and we realized there are columns which null values are high (See figure 1), so we decide to remove those columns and only work with the columns that does not contains null values (See code 5).

The removed columns are: *"UnusedColumn", "LateAircraftDelay", "SecurityDelay", "NASDelay", "WeatherDelay", "CarrierDelay", "AirTime", "ActualElapsedTime", "ArrivalDelay", "ArrivalTime", "TaxiIn", "WheelsOn"*.

Finally, we decided to work only with non diverted figths (See code 6). **After cleaning, our training data contains 6429338 rows.**

Listing 3: Code

```
renamed_columns = [
    "Date", "UniqueCarrier", "FlightNumber", "Origin", "Destination",
    "CRSDepTime", "DepartureTime", "DepartureDelay", "TaxiOut", "WheelsOff",
    "WheelsOn", "TaxiIn", "CRSArrivalTime", "ArrivalTime", "ArrivalDelay",
    "Cancelled", "CancellationCode", "Diverted", "CRSElapsedTime",
    "ActualElapsedTime", "AirTime", "Distance", "CarrierDelay",
    "WeatherDelay", "NASDelay", "SecurityDelay", "LateAircraftDelay",
    "UnusedColumn"
]

flights_2009 = flights_2009.toDF(*renamed_columns)

test_df = test_df.toDF(*renamed_columns)
```

Listing 4: Code

```
flights_2009 = flights_2009.withColumn("DayofWeek", F.dayofweek("Date")) \
                        .withColumn("Month", F.month("Date"))

test_df = test_df.withColumn("DayofWeek", F.dayofweek("Date")) \
                        .withColumn("Month", F.month("Date"))
```

```
test_df.select([
    F.count(F.when(F.col(c).isNull() | (F.isnan(c) if dict(test_df.dtypes)[c] in ('double', 'float') else F.lit(False)), c)).alias(c)
    for c in test_df.columns
]).show()
```

```
|Date|UniqueCarrier|FlightNumber|Origin|Destination|CRSDepTime|DepartureTime|DepartureDelay|TaxiOut|WheelsOff|WheelsOn|TaxiIn|CRSArrivalTime|ArrivalTime|ArrivalDelay|Cancelled|CancellationCode|Diverted|CRSElapsedTime|ActualElapsedTime|AirTime|Distance|CarrierDelay|WeatherDelay|NASDelay|SecurityDelay|LateAircraftDelay|UnusedColumn|DayofWeek|Month|
|   0|            0|           0|     0|          0|         0|       108777|        108777| 111949|   111949|  116060|116060|             0|     116060|      128729|        0|         6336862|       0|            17|           128729| 128729|       0|     5275233|     5275233| 5275233|      5275233|          6450117|           0|        0|    0|
```

Figure 1: Empty/null columns

Listing 5: Code

```
# Remove the columns with NULL values
flights_2009 = flights_2009
        .drop("UnusedColumn", "LateAircraftDelay", ... , "TaxiIn", "WheelsOn")

test_df =    test_df
        .drop("UnusedColumn", "LateAircraftDelay", ... , "TaxiIn", "WheelsOn")
```

Listing 6: Code

```
#Filter out Diverted = 1
flights_2009 = (
        flights_2009
        .filter(F.col("Diverted") != 1)
        .drop("Diverted")
)
test_df = (
        test_df
        .filter(F.col("Diverted") != 1)
        .drop("Diverted")
)
```

Exploratory Analysis (2 pts) 3

In our exploratory analysis, we found that **WN** is by far the the top 1 carrier across the top 10 with more than twice as flights as the second position ((See figure 2 and 3).

```
# Top 10 carriers
print("\n Top-10 carriers by flight count (2009):")
(flights_2009.groupBy("UniqueCarrier")
        .count()
        .orderBy(F.desc("count"))
        .limit(10)
        .show())
```

```
 Top-10 carriers by flight count (2009):
+-------------+-------+
|UniqueCarrier|  count|
+-------------+-------+
|           WN|1127045|
|           AA| 548194|
|           OO| 544843|
|           MQ| 434577|
|           DL| 424982|
|           US| 411274|
|           UA| 375501|
|           XE| 308340|
|           EV| 297874|
|           NW| 291856|
+-------------+-------+
```

Figure 2: Top 10 carriers

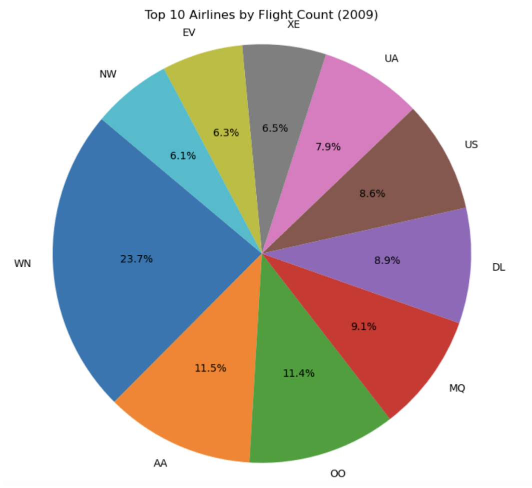

Figure 3: Top 10 carriers

From all the cancellation reasons, we found that **security** is the less probable reason, meanwhile **carrier** and **wheatear** are proportional (See figure 4 and 5).

```
# Cancellation reasons
print("\n Cancellation reasons count (A=carrier, B=weather, C=NAS, D=security):")
(flights_2009.filter(F.col("Cancelled") == 1)
        .groupBy("CancellationCode")
        .count()
        .orderBy(F.desc("count"))
        .show())
```

```
 Cancellation reasons count (A=carrier, B=weather, C=NAS, D=security):
+----------------+-----+
|CancellationCode|count|
+----------------+-----+
|               B|36651|
|               A|35568|
|               C|14799|
|               D|   20|
+----------------+-----+
```

Figure 4: Cancellation reasons

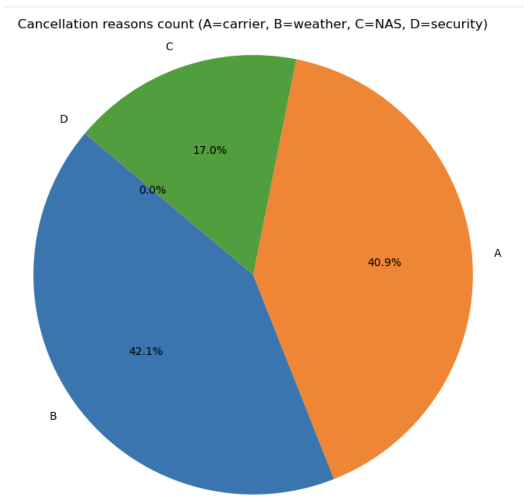

Figure 5: Cancellation reasons

Finally, we check the class balance distribution regarding wether a flight was canceled. We found that there is a huge rate (72.69%). See figure 6.

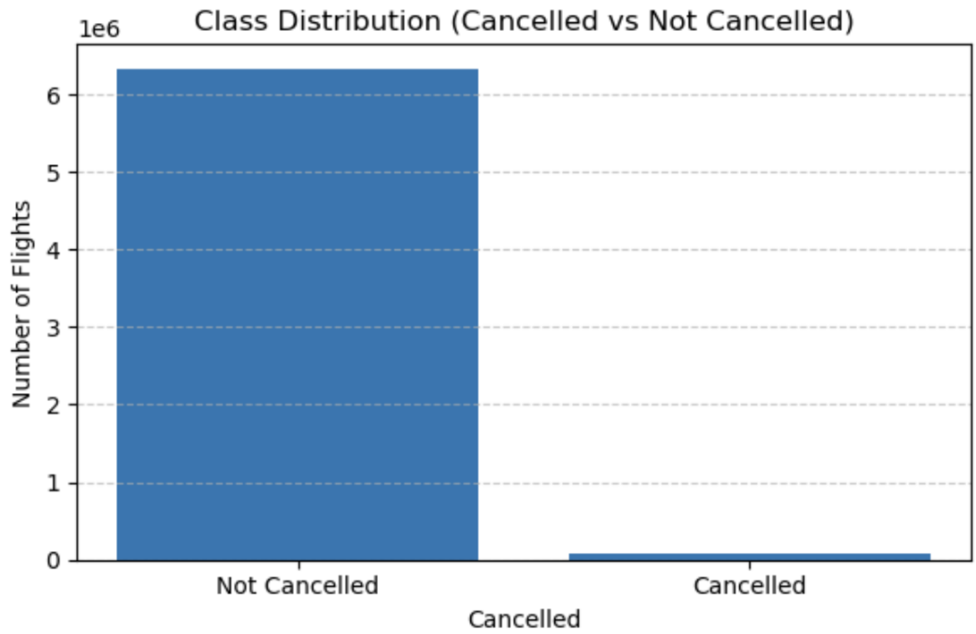

Figure 6: class balance distribution

Feature Engineering (3 pts) 4

We performed categorical feature processing with StringIndexer + OneHotEncoder on our categorical columns:

- Origin
- Destination
- CRSDepTime
- DayofWeek
- Month

Added stages for categorical features:

- StringIndexer (Origin - Origin_Index)
- OneHotEncoder (Origin_Index - Origin_Vec)
- StringIndexer (Destination - Destination_Index)
- OneHotEncoder (Destination_Index -Destination_Vec)
- StringIndexer (CRSDepTime - CRSDepTime_Index)
- OneHotEncoder (CRSDepTime_Index - CRSDepTime_Vec)

- StringIndexer (DayofWeek - DayofWeek_Index)
- OneHotEncoder (DayofWeek_Index - DayofWeek_Vec)
- StringIndexer (Month - Month_Index)
- OneHotEncoder (Month_Index - Month_Vec)

Using **StringIndexer** we also add the string index for the column **Cancelled**.

Finally we create features from our numerical rows using **VectorAssembler**, those columns are:

- CRSArrivalTime
- CRSElapsedTime
- Distance

Modeling (5 pts) 5

First, we split the training data into training (70%) and test (30%). Then, we train 4 models which their respective params grid builder:

- **LogisticRegression**: $lr.regParam \in [0.01, 0.1, 1.0]$
- **DecisionTreeClassifier**: $lr.regParam \in [0.01, 0.1, 1.0]$
- **RandomForestClassifier**: $rf.numTrees \in [20, 50, 100]$
- **GBTClassifier**: $gbt.maxIter \in [10, 20, 30]$

BinaryClassificationEvaluator with *areaUnderROC* and **MulticlassClassificationEvaluator** with *accuracy* were our evaluators.

Once models and evaluators were defined, we iterate over each model and their respective params to perform a Cross validation and optain the best model with the best params configuration (See code 7).

Listing 7: Code

```
cv_models   = {}
cv_best_auc = {}

for name, clf in models.items():
    cv = CrossValidator(
        estimator=clf,
        estimatorParamMaps=param_grids[name],
        evaluator=binary_eval,
        numFolds=3,
        seed=42
    )
    print(f" Running CV for {name}      ")
    cv_model = cv.fit(train_data)
# train_data already has features & label
    cv_models[name]      = cv_model
    cv_best_auc[name] = max(cv_model.avgMetrics)
    print(f"   {name} best CV AUC = {cv_best_auc[name]:.4f}")
```

During our test, Linear regression shows the best area under the curve with **0.7422**, followed by random Gradient Boosted Tree **0.7139** and Random Forest **0.5963**. The results could be analyzed in the next table:

Model	**Accuracy**	**AUC**
Logistic Regression	0.9864	0.7428
Decision Tree	0.9864	0.5000
Random Forest	0.9864	0.5963
Gradient Boosted Tree	0.9864	0.7139

Table 1: Performance metrics for different models

Explainability (1 pts) 6

Base on our analyzes, Month and destination are the two main features being February and June the two most important, and BOSNIA AND HERZEGOVINA airport as the main destination.

Figure 7 illustrate the top features.

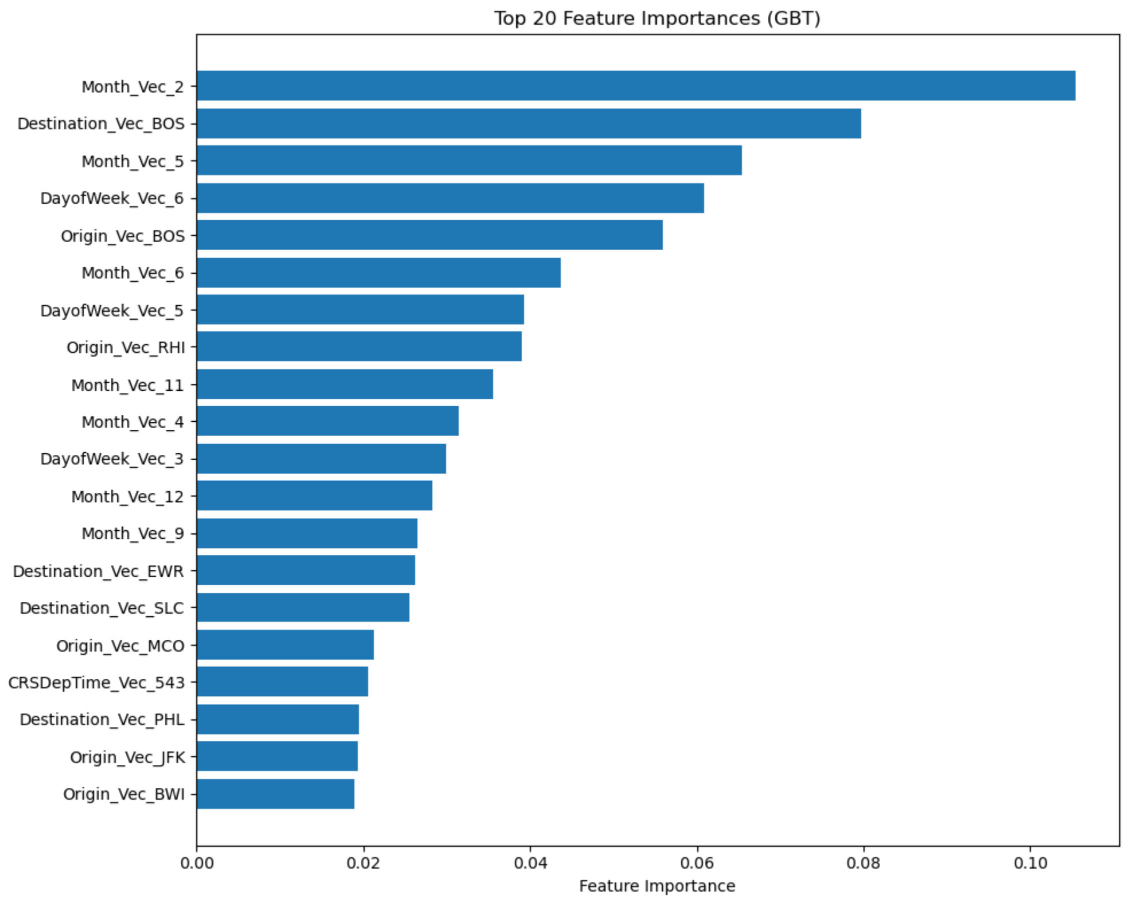

Figure 7: Top features

Model Persistence and Inference (2 pts) 7

Finally, the next code was used to save the model whose performance on 2010 Data was **AUC = 0.6403**, **Accuracy = 0.9824**.

Listing 8: Code

```
best_tree_model.write().overwrite().save("./best_model/")
preds2010 = best_tree_model.transform(test_df_2010)
auc2010   = binary_eval.evaluate(preds2010)
acc2010   = multi_eval.evaluate(preds2010)
print(f"\n2010 Data --> AUC = {auc2010:.4f}, Accuracy = {acc2010:.4f}")
```