CMPT 732, Fall 2023 - Programming for Big Data I

Project Title: Airline Delays Analysis and Prediction

Contributors: Zizheng Que, Lingyun Li, Shan Chen, Yangyang Jiang

Project Link

1) Problem Definition

We're tackling the common issue of flight delays by predicting them using airport locations and weather data. Our approach includes collecting and integrating weather and flight delay data to understand delay-causing factors. Utilizing machine learning, we've developed models for real-time delay predictions. Our AWS-hosted website, supported by a Cassandra database, enables users to input data for delay forecasts. Our challenges included collecting data, analyzing over 50 attributes, deploying front and back-end on the cloud, and selecting a cloud database.

2) Methodology

For our project, we employed a comprehensive approach to data processing, analysis, and web development. We utilized **Spark** and **Pandas** for data processing and analysis, ensuring efficient handling of large datasets. The backend was developed using **Python Flask**, integrated with **Pandas** and **Sklearn** for data processing. We further enhanced the user experience by designing an interactive frontend using **React** and **JavaScript**, enriched with dynamic charts created through **Chart.js** and **D3.js , along with an AWS Cassandra database** to enhances our system's ability to scale and handle multiple tasks at once.

During the prediction phase, we leveraged the power of **Scikit-learn** to implement various machine learning models. To tackle the challenge of class imbalance, we incorporated oversampling techniques from the **imbalanced-learn** library, enhancing our model's accuracy and reliability.

3) Data Processing

In our Data Processing section, we describe the preparation of our data. We began by collecting flight delay data from the Bureau of Transportation Statistics, offering insights on airport-specific delays. Weather information from the National Centers for Environmental Informationwas then integrated, providing context on flight times.

We prioritized identifying key attributes influencing flight delays and rectified any data anomalies for precision. The culmination of our process involved merging flight delay and weather data by aligning airport names and dates, a step essential for enhancing our analysis's computational efficiency. This method streamlined our approach, ensuring a focused and effective analysis.

4) Data Analysis

Following the data cleaning process, we conducted a comprehensive analysis of factors influencing airline delays, ultimately identifying **20 key factors** for further examination. These factors were categorized into **five dimensions: date, depart time, location, continuous numerical weather conditions, and categorical numerical weather conditions.** We utilized **PySpark** to extract these factors, followed by various transformations and analyses, leading to the **creation of 31 CSV files** for subsequent visualization purposes. Initially, we separately imported data from the years 2020 to 2022, storing them in **three distinct dataframes** for independent analysis specific to each year. Subsequently, we employed the **union method** to amalgamate these three dataframes, facilitating a comprehensive analysis encompassing the entire data set. Then, we filtered the dataset to include only flights with a delay time greater than zero. During factor analysis, any rows with null values for a specific factor were removed. The analysis of airline delay frequency and average delay duration was conducted across the following five dimensions:

4.1. Impact of Different Dates on Flight Delays: By grouping data by **"YEAR", "MONTH", and "DAY_OF_WEEK"**, and then aggregating them, we obtained the count for **"num_delays"** and the average for **"avg_delay_time"**. The results revealed a yearly increase in both delay frequency and duration from 2020 to 2022. The months of June to August showed higher delays compared to other months in the last two years. Weekday analysis indicated a slight increase in delay duration approaching weekends, though not significantly. We defined functions **get_month_name** and **get_day_of_week_name** for future visualization purposes.

4.2. Effect of Departure Time on Flight Delays: Grouping by **"DEP_TIME"** showed a **significant impact of departure time on flight delays**, with a **consistent pattern** observed yearly. From midnight to 4 AM, flight delays were fewer, likely due to fewer scheduled flights. However, the average delay duration during these hours was significantly higher than during daytime hours, a factor expected to be crucial in subsequent delay predictions.

4.3. Location-Based Analysis of Flight Delays: Grouping by **state** and **city**, we found that Texas and California had the highest number of delays. Wyoming had the highest average delay duration, and Hawaii had the lowest. From a city perspective, ATL had the highest delay frequency, and PSE the lowest. EKO and ITO had the highest and lowest average delay durations, respectively. We also analyzed the **top ten states** and **top twenty cities** in terms of delay frequency, using **broadcast join** methods with original data to examine their monthly delay patterns. Additionally, **we divided elevation into 24 intervals**, finding that the highest average delay (93.30 minutes) occurred at elevations between 1500 and 1600 meters, and the lowest (33.97 minutes) between 600 and 700 meters.

4.4. Impact of Continuous Numerical Weather Conditions on Flight Delays: We analyzed delays based on **average temperature, dew point, visibility, wind speed, maximum sustained wind speed, precipitation, and snow depth**. Each factor was divided into intervals, and data was grouped accordingly. Temperature was converted from Fahrenheit to Celsius for clarity. Visibility was categorized based on National Weather Service standards: **low (under 0.4 miles), moderate (0.4 - 1 miles), and high (over 1 miles)**. The analysis revealed that extreme weather conditions showed the longest delay durations, while milder conditions correlated with shorter delays. In addition, we also examined delays by different states and cities under different visibility categories.

4.5. Impact of Categorical Numerical Weather Conditions on Flight Delays: Analyzing delays under **severe weather conditions like fog, snow, hail, thunderstorms, and tornadoes**, we found that the presence of such weather (indicated by 1 for presence and 0 for absence) increased average delay durations significantly, ranging from 24.4% to 64.5%.

5) Data Visualization

To visualize our data analysis more clearly, we constructed **line, bar and geographical charts** to show the relationship between these factors and flight delay times. To make the correlations clearer, we used different types of charts. For time factors such as year, month, and hour, we used **line graphs** to better differentiate the impact of various time periods on flight delays. For geographical factors like airport state location, we applied **geographical map** and marked specific data for each state. For weather factors such as temperature, humidity, and extreme weather conditions, we used **bar charts** to better demonstrate how different weather conditions affect flight time delays.

Additionally, to make our analysis more directly accessible and understandable for users, we transformed our analytical charts and predictive models into a **web browser** and **deployed them on EC2** for easy user viewing and interaction. We used **Python Flask** to create the backend framework and employed **Pandas and Sklearn** for backend computational logic. Simultaneously, we set up a **Cassandra database** for flexible access to updated data. On the frontend, we used **React and JavaScript** for the basic framework and incorporated **Chart.js and D3.js** for constructing interactive analytical charts. The interactive functionality of Chart.js allow users to clearly understand the charts. We also added **MaterialUI** and basic **CSS** to enhance the UI.

We also developed a model to predict flight delays. Based on the accuracy of the predictions and other evaluation criteria, we chose the **MLP (Multilayer Perceptron) model** as our prediction model. This model is fully deployed on **EC2**. After users submit the airport data through the **prediction form** on our website, it will quickly return the prediction results to inform users about potential flight delays time.

6) Flight-Status Prediction

6.1. Defining Problem: For practical application, we aim to make predictions based on flight and weather information input by users on a webpage. Therefore, we selected a subset of easily accessible weather information and some flight featrues that can be easily obtained from the boarding pass as input features. In the output section, we defined it as a **binary classification problem**: class 0 for normal flights (departing on time or delayed by less than 15 minutes) and class 1 for abnormal flights (delayed by more than 15 minutes or canceled).

6.2. Data Preprocessing and Feature Engineering: We used **Scikit-learn** and **Pandas** for feature engineering and implementing our machine learning model.

1. Necessary Imputation for Missing Data: In the dataset, flights labeled as canceled have missing values in the departure time column. We used the **random sampling** method to fill missing values by selecting departure times from non-canceled flights of the same year.

2. Handling Imbalanced Data: The dataset exhibited a severe class imbalance, with class 0 rows dominating. To mitigate this, we employed **oversampling** on class 1 data in the training set to balance the quantities of both classes.

3. Numerical Feature Standardization: Given the wide range and magnitude differences in numerical features (e.g., temperature, latitude, altitude). Thus, we normalized these features with employing StandardScaler to enhance the model's convergence speed and stability.

4. Handling Discrete Features:For features represented numerically but lacking clear numerical relationships (e.g., months), we used **OneHotEncoder** to convert these columns into one-hot encoding to avoid misleading the model.

6.3. Machine Learning Models:

Metrics: Common evaluation metrics for binary classification include accuracy, recall and so on**.** Here we use **Balanced F_1** score, the average of F_1 scores for both classes, was chosen as the final evaluation metric to address data imbalance.

Methods: We experimented with various machine learning models, comparing their performance on a subset of the dataset. This included **RidgeClassifier**, **K-Nearest Neighbors (KNN)**, **Decision Tree**, **Random Forest**, and **Multilayer Perceptron (MLP)**. RidgeClassifier performed poorly due to the non-linear relationship between input and output. KNN and Decision Tree underperformed compared to Random Forest and MLP. For KNN, we attribute this to the high dimensionality of the input data, resulting in sparse distribution in the high-dimensional space, negatively impacting distance metrics like Euclidean distance. Regarding Decision Tree, we believe its generalization ability for high-dimensional data is inferior to Random Forest and MLP. Random Forest and MLP exhibited similar performance, but considering the larger size and slower convergence speed of the Random Forest model, we ultimately chose MLP as our predictive model.

7) Database

We started by creating a unique keyspace on **AWS** for our **Cassandra** database. To connect our local setup to Cassandra, we used technologies like **SSLContext**, **Cluster**, and **PlainTextAuthProvider**. We managed to store 31 CSV files in our Cassandra database by executing queries.

Our setup allows us to add new data easily. Whenever we have new data, we can simply insert it into the corresponding table in our database. This is ideal for handling big data because Cassandra is well-suited for managing **large amounts of data** and high traffic, especially in a cloud environment Cloud services.

Moreover, our data has great **scalability** and can also perform **parallel processing**. Cassandra's **distributed architecture** allows for efficient **parallel computation**, which means we can handle multiple operations simultaneously.

Additionally, hosting both our backend and database on the cloud enhances the interaction. In our backend, we've enabled **real-time interaction** between user input and our Cassandra data. Specifically, when users input their requests on the frontend, our backend retrieves the relevant data using **CQL queries** from Cassandra and sends it back to the frontend.

Overall, our system's design on AWS harnesses the power of Cassandra for **big data managemen**t, scalability, user **interactions**, and parallel processing.

8) Problems and Challenges

Some of the Problems and challenges we encountered in the project are:

8.1 Code efficiency

During the data analyzing, we faced challenges due to the large volume of data. By applying techniques learned during this semester, such as **Cache, Join, and Broadcast**, we improved code quality and PySpark efficiency. We cached data for reusing intermediate results across multiple actions. When we cached a DataFrame, PySpark kept the data in memory, which speeded up subsequent computations using this data. We **strategically cached specific dataframes at appropriate stages of the analysis**. After considering the size and distribution of our data, we used **Broadcast Joins** from the **pyspark.sql.functions** when we're joining a significant larger initial DataFrame with a smaller one. By broadcasting the smaller DataFrame, PySpark sent it to all the nodes in the cluster, so it was available in-memory, reducing the need for shuffling data across the network.

8.2 Database Scalability

Our original data is organized by year, and our model uses data from three years. To consider future scalability, such as extending the data from 2000 to 2023 and beyond, storing data in CSV files may not be very convenient. By using a database, we can enhance data scalability. New data can be easily inserted into the corresponding tables, making it well-suited for handling large datasets. Additionally, since our backend is deployed in the cloud, placing the database in the cloud enhances interaction between the backend and the database.

8.3 Bulk CSV Data Ingestion

We needed to import nearly 31 tables into AWS in bulk. These tables were not standardized, with each having its own format and attributes. Furthermore, every piece of information in the CSV files was treated as a string, whereas Cassandra requires specific data types for each column. To address this issue, I wrote a function that automatically identified the header of each table, checked and input the data types for each column. This allowed us to automate the handling of table-specific information in bulk. Our tables had various PRIMARY KEY configurations, some with a single column and others with multiple columns. We determined the list of PRIMARY KEY columns for each table based on its specific requirements, enabling us to automate this process. During debugging, I encountered issues where data insertion failed due to the table not being found. Upon revisiting the AWS keyspace webpage, I discovered that while the table creation code had been executed in Python, AWS required some time to create the tables. To address this, I added a time delay (timesleep) in the code to allow AWS sufficient time to respond. Similarly, during bulk data ingestion, I added timesleep between each row insertion to ensure AWS had adequate response time.

8.4 Training Models on Imbalanced Dataae

During data analysis, we observed a significant imbalance in the dataset, where rows belonging to class 0 constituted the vast majority. Taking the test set as an example, class 0 accounted for 85.82% of the total data. Such a severe data imbalance can lead to a substantial bias in the model optimization towards class 0, thereby compromising the model's practicality (as users are more concerned with class 1, representing abnormal flight situations) in real-world scenarios. Therefore, exclusively for the training set, we employed the over_sampling() method from the **imbalanced-learn** library to oversample data from class 1, ensuring a balanced representation of both classes. After this adjustment, although the overall F1 score decreased from 0.82 to 0.75, the **Balanced F1 score**, which is our focus, increased from 0.57 to 0.63.   While the numerical values of the metrics may not appear exceptionally high, considering that aircraft delays are influenced by factors beyond just weather conditions, this improvement is noteworthy.

9) Project Summary

Category	Points
Getting the data: Download raw data from two sources and join them with pandas.	2
ETL: Understanding the data, identifying 20 key factors for analyzing, and filtering irrelevant data. PySpark job for Extract-Transform-Load. Improving code efficiency by using Cache, Join and Broadcast.	3
Problem: We aim to analyze the relationship between flight delays and airport weather conditions through data analysis, which holds practical significance. Building upon this, we define the task of predicting flight delays as a binary classification problem and employ machine learning models to address it, thereby facilitating travel for individuals. Subsequently, we develop a website hosting visualized analysis results and prediction interfaces for reference and used by the public.	2
Algorithmic work: Using Scikit-learn, we employed various machine learning models to predict the probability of flight delay. We mitigated the significant class imbalance by utilizing the Imbalanced-learn library.	2
Bigness/parallelization: Cache, Broadcast, Cassandra database. Our project utilizes the Cassandra database, which, thanks to its distributed architecture and parallel	2

capabilities, allows us to scale as the data volume increases.	
UI: Use React, Javascript and material UI to create the interactive UI including dynamic charts and submission form, and Hosting the website on EC2 using serve, PM2, gunicorn and nginx.	3
Visualization: Use Chart.js and D3.js to plot bar charts, line charts and geographical map charts. Reveal the relationship between the average flight time delay and other factors causing them.	2
Technologies: Spark, Pandas, Flask, React, Javascript, Sklearn, imbalanced-learn, Chart.js, D3.js, deploy on EC2 using serve, PM2, gunicorn and nginx, Cassandra; Multilayer Perceptron (MLP), Random Forest.	4
Total	**20**

10) References

https://www.kaggle.com/datasets/threnjen/2019-airline-delays-and-cancellations

https://www.ncei.noaa.gov/

https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FGK

11) Results

We've created a website where users can interactively input information to predict flight delays, detailed in sections 5) Data Visualization and 6) Flight-Status Prediction. We used the AWS Cassandra database for enhanced data scalability and parallelism, as explained in section 7) Database. Here is our website's URL, and we welcome everyone to use it.