PROBLEM INTRO

The FAA annually handles over 16 million flights out of 20,000 airports (about 5K public commercial and 15K private) to service 1 billion passengers and 44 trillion tons of freight. This translates to about 44,000 flights operating daily serving about 3 million passengers. Given the expanse and complexity of operations, delays in airline and airport operations are therefore inevitable. As of 2018, almost 1 in 5 US flights suffered a delay.

However delays have been on the increase to the extent that they are amounting to billions of dollars of economic loss. As of 2019, worldwide costs due to airline delays were estimated at $60 billion, with over a half incurred due to US flights. Additionally, US aviation reported a fuel wastage of 740 million gallons and CO_2 emissions amounting to 7.1 million tons. Clearly this problem has a huge impact in several areas and on several stakeholders.

Economic losses due to delays impact airlines and airports(@75$/min) directly and business travelers and vacationers indirectly in terms of lost productivity and leisure time – not to mention all the supporting travel service companies.

The latest forecast from the FAA indicates an increase of 65% in revenue passenger miles by 2037, therefore finding a practical and scalable solution for mitigating delays is imperative.

Efforts to address this problem so far include expanding aviation infrastructure, improvements to existing air traffic management systems and congestion pricing, all of which are proving to be insufficient in addressing the problem at the scale in which it is growing.

Computationally speaking , researchers in the past have taken several approaches and have applied various mathematical techniques but size of the problem and lack of computational power remained a huge hurldle. but in recent years, improvements in big data technology and machine learning are allowing researchers to look at developing scalable algorithms that can predict delays at an individual flight level with the help of openly sourced data. Such solutions are expected to work alongside existing Air Traffic Management Systems to improve air space management, and also help airlines, airports and passengers respond to delays and mitigate losses proactively.

So if you scroll down….for this project, our team set out to develop and implement a scalable machine learning model using individual flight and location specific weather data to predict departure delays that exceed 15 minutes. The target precision for our model was set at 85% with a run-time goal of 30 minutes - which would give us a sense of whether this model would meet the time constraints for it to be converted into a viable product.

DATA SOURCES

- Our primary data source were the monthly reports of Carrier-on-Time Performance
- Our secondary data source was weather data - which comprised reports of several weather related measurements taken hourly at various weather stations.
- Our ancillary data consisted of data that provided airport code information and a list of holidays that were scraped from the web.

INITIAL ANALYSIS

As a part of our initial analysis, we studied the
- distribution of delays by duration and
- distribution of delays by delay type, and more specifically, the contribution of weather to these delays.

Looking at Delays by Duration

Most delays fell into the 15-29 min therefore being able to detect these at the very least would certainly be useful

Next Looking at Delays by Delay Classification

Delays could occur due to several reasons. What we found from the data was interesting:

1 - most departure delays occurred due to late arriving flights on the previous leg. and

2 – contrary to what we had anticipated, delays due to weather did not constitute a majority

Upon researching further we learned that Total weather delays = extreme weather + NAS + % of late arrivals
And that weather's share towards delays was far greater.

This helped us establish perspective on a couple things:
1 - to avoid a pitfall initially of eliminating records that were in some instances not coded as delayed due to weather
2 - to not place undue importance on weather measurements as being primary predictors of delay.
3 - in the event that the join with weather data becomes expensive, to determine whether and how much of weather related data information needs to be brought in without negatively impacting our model performance

Therefore armed with this information, we set out to design our solution in 3 phases
- Data ingestion
- Feature engineering
- Modeling

WEATHER

Our initial exploratory analysis and further research around delays revealed that while flight delays vary by carrier, day of week , month of year and distance, the variation of delays attributed due to weather were not completely captured by the WEATHER_DELAY field as we first imagined.

It was useful to learn that the WEATHER_DELAY field only captures about 6% of the total weather related delays while the remaining are actually included in the NAS_DELAY and LATE_AIRCRAFT_DELAY.

Prior research in the area and information by BTS (Bureau of Transportation Statistics) confirmed that weather delays may constitute as much as 40% of the total delays and as much as 20% of delays are caused by Late Arriving aircrafts.

This knowledge helped us establish perspective on a few things:

- to avoid the potential pitfall of eliminating records that were in some instances not coded as delayed due to weather for training
- to place undue importance on weather measurements as being the dominating predictor of delay.
- to experiment and compare the accuracy of different models, one trained on features that include weather related data Vs another trained on features that include just flight information.
- while weather data is important, since joins between massive tables could be expensive, we could consider bringing in a subset of columns required for the success of our model without negatively impacting the performance of our data pipeline.

DATA PIPELINE

For the purpose of organizing our data engineering, we adopted a model like the one Databricks documents, using a metaphor of bronze, silver and gold datasets. We use the terms in this way:

Bronze - thin facade over the raw data sources to make them usable in spark.

Silver - processed, regularized, joined datasets

Gold finished data ready for ML modeling

This helps divide tasks and responsibilities. EDA and data regularization (normalization, correction, imputation) happen in Silver. Feature engineering also happens in Silver, although high-level feature engineering such as feature selection and PCA happen in Gold. We found it useful to introduce sub-groups to Silver: "clean" tables have data regularization. Pure silver tables may be the result of joins and may be enriched with derived features.

DATA CLEANING

Once we figured these critical aspects we progressed with data import, cleansing, transforming, and joining our airlines and weather data.

As a part of data import, we imported the following raw data and stored it as our Bronze Data sets: Airports.data Holidays data Airlines data Weather data

As a part of data cleansing, we: eliminated duplicates imputed nulls to zeroes wherever applicable handled invalid time zones by setting them to Eastern Assessed the impact of invalid airport codes and airport codes that could not be mapped to weather stations. There were two flight origins (local/municipal airports) in our training data that could not be mapped to weather stations. These accounted to <1% of total flights. Given the characteristics, we decided to drop these from our training set downstream.

As a part of data transformations, we: converted the arrival and departure times of flights to UTC generated hourly time buckets parsed compound weather data fields

TIME NORMALIZATION

Weather data has timestamps which are unambiguous representations of instants, independent of time zone, but times in the airlines dataset are awkwardly represented as decimal numbers and textual days: departure times are in the time zone of the origin airport and arrivals are in the time zone of the destination. We chose to normalize all these times to timestamps by converting them to UTC. We did this by using spark builtin conversion, but this concealed some details, such as daylight vs. standard time.

We used a publicly available airports database to get the time zone of airports and to assist in matching airline codes (IATA vs ICAO). This dataset covers almost all the airports that are referenced in the "airlines" (flights) dataset, but we needed to impute values for some missing airports.

TIME BUCKETS

Since there are many events in the system that we may want to join and aggregate, and these may all occur at slightly different timestamps, we introduced an idea of time "buckets". These are time intervals into which different events can be aggregated and by which aggregations can be joined. We chose hour-wide buckets, similar to the "TIME_BLK" fields of the airlines data, but unified to UTC time. These time buckets were to be used only as handles for data engineering and aggregation; they are not "features" in the ML sense - whereas the local TIME_BLK may be a feature that has predictive value.

We believed it is crucial to consider only data that was available at the prediction time, which is two hours before the flight departure time. We added a prediction time bucket to each flight, which we used as the join key to the weather data.

DATA JOINS

As a part of joins, we

- First joined airlines data with airports. The ORIGIN and DEST fields in airlines data mapped to the IATA field in the airports table
- Next we joined weather data to airports data. The CALL_SIGN in weather data mapped to the ICAO of the airports data
- Our big join consisted of joining the results of the above two joins.

The results of these joins were saved as our Silver Data.

- We then joined on the aggregate delay data. This data was much smaller, since it has at most one record for each (airport, hour)

AIRPORT BUSYNESS AGGREGATION

We speculated that previous delays at an airport could contribute to further delays, so we developed aggregations that express delays, as evidenced by actual flights delayed, and planned busyness, as expressed by scheduled activity. Although these two kinds of measures are both aggregated by bucket for each airport, we were very cautious about how we join these aggregates. Scheduled activity in a given bucket can be joined with flights that depart in that bucket, since schedules are known in advance, but aggregate actual delay must be associated with the prediction time bucket, since at prediction time, we can only predict on the basis of the delays that have happened before prediction time. Moreover, since the full actual delays in minutes are not known until the flight actually takes off, we must aggregate actual delays by actual departure time. This might cause bad predictiveness in the case of very long delayed flights.

FEATURE ENGINEERING

Input feature attributes for our model were based mostly on intuition and work previously undertaken by researchers. We classified features into the following broad categories:

Spatial characteristics: origin, destination

Temporal characteristics: month, day of week, time of day

Flight Performance characteristics: planned arrival, planned departure schedules

Weather characteristics: represent external and environmental conditions

State of the System characteristics: cumulative delays by category, number of flights departing in the hour, runway performance (taxi in/out) and late arrivals

Most were readily available in the Silver data while the remaining new features were aggregated separately and joined with the Silver data. As previously pointed out, time buckets were used to both generate these counts as well as append them as features of each of the flights.

We also noted that correlated and irrelevant features may provide model overfitting and decrease prediction performance, and analysing feature importance may enable us to discriminate and study the most impactful features. Although we did not have a chance to implement this in the current approach, we plan to do so in our next iteration

Next, all categorical features were identified and converted to strings and all numeric variables were converted to doubles and the flights along with selected features were saved as our GOLD data.

MODEL EXPLORATION

We explored Logistic Regression (LR), Random Forest (RF) and Gradient Boosted Trees (GBT), and finally chose LR and GBT.

Logistic regression returns probabilities of belonging to class 0 or class 1, hence one could set an appropriate threshold that would maximize precision (as opposed to recall). For instance, one could set the decision threshold at a probability of 0.9 for the label to be deemed positive (1, or delay in our case). While we expect logistic regression to be decently performant, it could suffer greatly from overfitting, and hence result in a high variance.

By choosing ensemble methods like Random Forests and Gradient Boosted Trees, we can trade bias for variance (that is, we want a model that is reasonably robust to incoming/updated training data). While we really wanted to implement LightGBM and XGBoost, these were not readily available to us - the former could not be loaded onto the cluster due to issues with the third party Microsoft Azure library, while the latter was available only in Scala. As a compromise, we chose GBT.

In order to further improve model performance, we used stacking by taking the predictions of logistic regression and GBT and passing them through another meta-estimator GBT. We found that the performance of the stacked model was the best overall, followed by GBT and then LR.

- Tried Logistic Regression (LR), Random Forest (RF) and Gradient Boosted Trees (GBT)
- **Trade bias for variance using ensemble methods** like GBTs
- LightGBM and XGBoost were not readily available to us
- In order to further improve model performance, we used **stacking of LR and GBT with GBT as meta-estimator**

GRADIENT BOOSTED TREES

GBT is an effective ensemble learning algorithm based on the idea of boosting. It’s easier to explain for the case of regression, than for classification. A first decision tree regressor is built. Then, its residuals are used to train a second tree (after multiplying the residuals with a learning rate). Then a third tree is built based on the new residuals, and so on. Boosting comes from the learning rate, which is a hyperparameter, and

helps to reduce bias. Thus GBT can greatly overfit leading to high variance, and therefore the depth of trees as well as the total number of boosted trees are used as hyperparameters to improve the variance. GBT for classification works on the same principles, but deals with log-likelihood maximization instead of residual minimization.

Categorical features were one-hot encoded, whereby they were first indexed and then converted to n dimensional vectors (for a category consisting of n levels).

We experimented with standardization techniques using StandardScaler and RobustScaler, but found they did not really help much. This may be because use of feature scaling in the algorithms we chose (LR and GBT) were already robust to features lying in different ranges.

We chose 2015-2018 as our training dataset, and the 2019 data for testing. We considered large periods of time to ensure that the inferred model is able to predict delays due to almost every condition except for rare events not captured in training data.

Since the flights with DEP_DEL15 flag of 1 were only ~18% of the dataset, there was considerable data imbalance, and hence there was a great chance of getting the minor class predictions incorrect (due to the low training examples for that class). To overcome this, we utilized class weights option in the logistic regression, but saw only a very small improvement (suggesting that the variance in our models was very small already). We looked to implement SMOTE, but found it was not readily available in PySpark.

We used GridSearch for hyperparameter tuning, and implemented TrainingValidationSplit (TSV) with 0.75/0.25 ratio during initial stages of our model building before switching to CrossValidation (5-fold CV). We found that TSV gave very similar prediction scores as 5-fold CV, with the added advantage of being able to run the code in practical times (~3hrs compared to >10 hrs).

While our goal was to improve precision, we chose areaUnderPR as our metric of choice for optimization of the tradeoff between precision and recall. One can get a Precision of perfect 1.0 by predicting 0 for all examples! Clearly, this is undesired. For an imbalanced dataset as in our case, an F-beta score may be more appropriate as it provides a configurable mixture of precision vs recall . Specifically, F0.5 score with more emphasis on precision would be a metric of choice if our business requirement was to minimize false positives. However, for practical use, F1 score with balance weightage given to precision and recall may be a healthy compromise, and is one of the key scores that we rely on.

https://machinelearningmastery.com/fbeta-measure-for-machine-learning/#:~:text=The%20choice%20of%20the%20beta,measure%20or%20the%20F1%2Dscore.

MODEL RESULTS

- 5-fold **CrossValidation** did not show any significant improvement over **TrainValidationSplit** in 3:1 ratio. Summary of both are shown below.
  - Training time of CV was signficantly longer, understandably
- **weightedPrecision** and **F1** scores considered for model performance evaluation
- **weightedPrecision improved from LR-->GBT-->Stacking**
- **F1 improved from LR-->GBT** (stacking did not show further improvement)

CONCLUSIONS

Join Performance and Scalability

The biggest join is between airlines (flights) and weather. We prepared for this by assigning each weather report to a time bucket and making sure to consider only the latest report in each bucket if there were multiple. Pyspark window functions were great for this. Window functions are very expressive and flexible and quite difficult to understand and test.

The scalability of this join was really not a problem, even with the largest datasets. However, correctness was difficult. We benefited by putting some strict assertions in our code to make sure that we had not dropped any data or inadvertently duplicated it.

Cloud Computing

Besides providing the necessary computing resources for our experiments, the cloud makes it possible for the proposed solution to be easily scalable. In fact, if the amount of data increases (e.g., by extending the analysis to many years of flight and weather data), the cloud can provide the required resources with a high level of elasticity, reliability, and scalability.

Solution Approach

The following are the overall **merits** of our approach:

- Many models do not account for weather, but we have a chance to now include weather, which has the potential to improve accuracy of predictions
- any models predict aggregate delays at major airports which are not specific enough for most stakeholders involved to take action. Delay predictions of individual flights may fill this gap.
- The pre-vision horizon for most models is 24 hrs, using weather, this can be upto 4-5 days in advance
- However we did note that most of the reasons for delays are stochastic phenomena which are difficult to predict timely and accurately.

While our current algorithm did not quite meet the goals that we initially established, we realized a few merits to this approach.

- Many models do not account for weather, but we have a chance to now include weather, which has the potential to improve accuracy of predictions
- The pre-vision horizon for most models is 24 hrs, using weather, this can be upto 4-5 days in advance
- any models predict aggregate delays at major airports which are not specific enough for most stakeholders involved to take action. Delay predictions of individual flights may fill this gap.

Our **future work** could include:

- Data balancing using sophisticated techniques such as SMOTE
- Multi-class classification based on delay duration group - we imagine that in addition to knowing that a flight would be delayed, it would perhaps be more helpful to know whether a flight would be delayed by 15-30 minutes vs 30-45min.
- Considering multiple time windows for weather related features for greater precision
- Implementing some sort of page rank for the airports to reflect its capacity to handle delays

Selected Applications of Course Concepts

Scalability: One of the first issues we confronted in this project is the sheer size of the dataset: over 300 million records of weather data and gigabytes of flight data. And while we were able to conduct our initial EDA and algorithm exploration on a much smaller subset, we needed to ensure every aspect of our approach was scalable to the entire dataset. We relied on the use of PySpark data structures such as DataFrames as well as keeping all our transformations and other operations on the Spark cluster of machines. This allowed us to scale from or first attempts in this project on the smaller dataset up to the complete version. Only our approach to EDA required a subtle change as we no longer run EDA on the entire dataset but instead on a 1% sample.

One Hot Encoding / Vector Embeddings: We wanted to train our LG model with certain categorical features such as Origin and Destination airports, an important determinant of departure delays as we showed in our EDA and feature selection sections. We employed helper functions VectorAssembler and RFormula provided by the ML Feature library in order to convert these columns from their original string data types into a series of columns that are one hot encoded to represent their respective airports. This transformation helps binarize the feature vector and make the model fit more efficient.

Model Complexity / bias variance tradeoff / regularization: If our model is too simple, then we would run the risk of underfitting with high bias. On the other hand, if the model is too complex with many features, then the number of features after OHE etc would explode leading to overfitting with high variance. This is bias-variance tradeoff, and can be partially averted by using ensemble methods like Random Forests, Gradient Boosted Trees etc. For non-ensemble algorithms like logistic regression, L1 and L2

regularization can be utilized. Very high L1 regularization would drive the weights to zero, and thus help to indirectly do feature selection. In our case, we implemented ElasticNet so we could tune the extent of L1 and L2 regularization.

Research Citations

- The Economic Cost of Airline Flight Delay, Everett B. Peterson, Kevin Neels, Nathan Barczi and Thea Graham,: Journal of Transport Economics and Policy , January 2013, Vol. 47, No. 1 (January 2013), pp. 107-121

- J. J. Rebollo and H. Balakrishnan. Characterization and prediction of air trac delays.Transportation Research Part C: Emerging Technologies, 44(Supplement C):231–241,July 2014

- Using Scalable Data Mining for Predicting Flight Delays, L. Belcastro, Domenico Talia, Fabrizio Marozzo, Paolo Trunfio, ACM Transactions on Intelligent Systems and Technology 8(1) · January 2016

- A Review on Flight Delay Prediction, Alice Sternberg, Jorge Soares, Diego Carvalho, Eduardo Ogasawara CEFET/RJ, Rio de Janeiro, Brazil, November 6, 2017

- Bad Weather and Flight Delays: The Impact of Sudden and Slow Onset Weather Events, Stefan Borsky, Christian Unterberger, Economics of Transportation 18(June):10-26 · March 2019

- A Methodology for Predicting Aggregate Flight Departure Delays in Airports Based on Supervised Learning, Bojia Ye, Bo Liu, Yong Tian, Lili Wan, MDPI Basel, Switzerland, April 2020