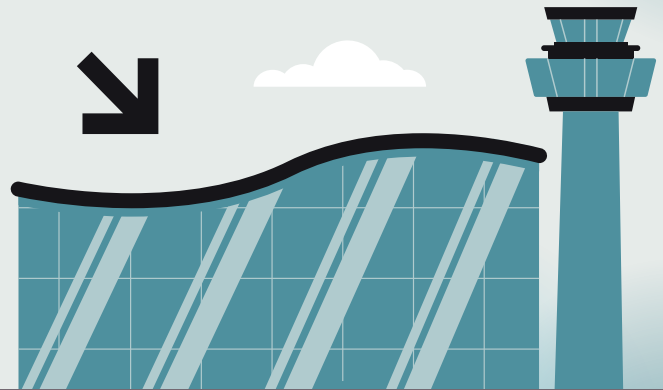


Allay Airway Delay!

Predicting Flight Delays to Reduce Wasted Customer Time

Section 4, Group 1, Team 13
Sparks and Stripes Forever

Nashat Cabral
Deanna Emery
Nina Huang
Ryan S. Wong



1

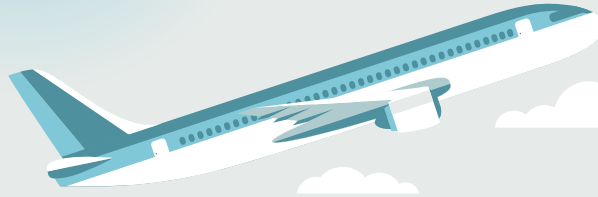
Nash 7

Hello! We are Team 13, also known as Sparks and Stripes Forever, and we are here to make flying a little less miserable for our users.

Link to notebook:

<https://adb-731998097721284.4.azuredatabricks.net/?o=731998097721284#notebook/632558266974488/command/1215577238246617>

The Project



Create a machine learning model for predicting which flights will be delayed. For airline travelers, knowing what flights are delayed allows them to better schedule their time and reduce wasted time.

**Delayed Flight = Flight Delay > 15 Minutes
OR Flight is Cancelled**

Nash 25 seconds

Delays are generally recognized as one of the worst parts of air travel. In 2007, travellers incurred nearly \$17 billion dollars worth of expenses due to delayed flights. As such, our goal is to use machine learning models and newly engineered features to predict whether or not a domestic flight will be delayed by 15 minutes or more given information from 2 hours prior to the scheduled departure time. Our aim is improve the experience of air travellers, by reducing the amount of time they waste waiting for a delayed flight.

Executive Summary

- Created a joined data set with over 40 million rows with 10 new, highly-predictive features.
 - Most influential features were previous flight delay, extreme weather, and departure hour indicators.
- Implemented a data pipeline to train, validate, and evaluate ML models.

Baseline Model

Logistic Regression

F0.5 = 0.20
(Primary Metric)

166 % Improvement
in predictive power
over baseline

Best Model

Gradient Boosted Tree

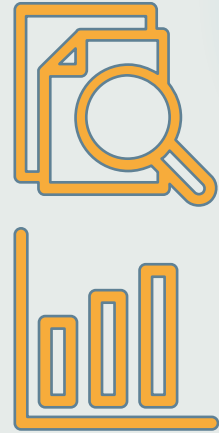
F0.5 = 0.52
(Primary Metric)

Nash 42

Building off of a simple model as a baseline, we were able to improve delay prediction performance by 166%, providing a powerful predictive tool that can help airline fliers avoid delays and wasting time. Throughout our analysis, we incorporated flight, weather, and weather station data, and developed several new, highly predictive features. We trained five different machine learning models using cross validation, and achieved best results from a Gradient Boosted Tree model, as measured by F0.5 and precision metrics. The most important features for this top-performing model were indicators for whether the flight was previously delayed or not, the hour of the flight's scheduled departure time, and the flight's airline carrier.

Data Set and EDA

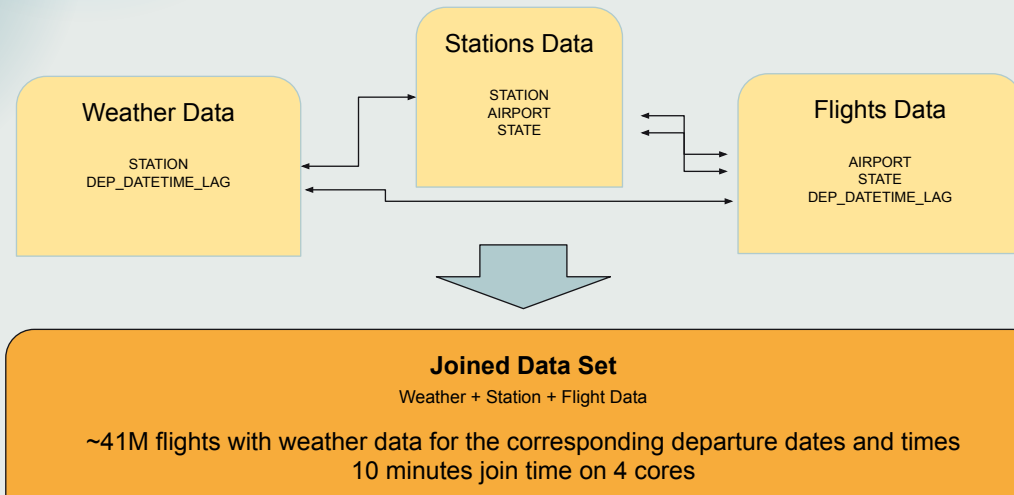
Airline Flights + Weather Stations + Weather Data



Deanna - 12s

For this analysis, our story begins with three data sets, including US domestic airline flight data for 2015-2021, weather data collected from hundreds of weather stations, and geographical data on the weather stations.

Joining Data Sets



5

Deanna - 30s

To join the data sources together, we connected the weather stations with the flights by associating airports with any nearby weather stations. We then joined the flights data onto the weather data based on the time of weather observation and the time **two** hours prior to the flight departure, to avoid data leakage. In instances where airports had multiple weather stations, we averaged a sample of their weather measurements together, allowing us to maximize our data coverage to account for stations with missing data. Finally, we converted all times to UTC to better track flights that cross time zones.

Joined Data EDA

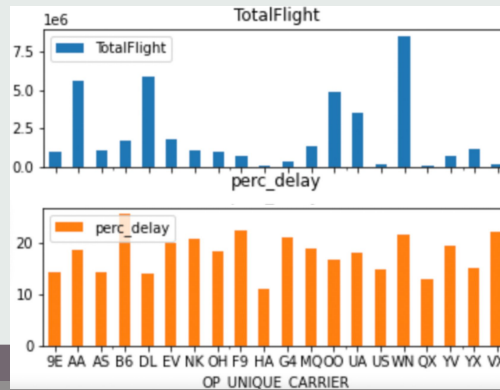
of rows in joined
data set

40,933,735

35 Million Non Delayed
6 Million Delayed

	DEP_DEL15
YEAR	-0.01
DEP_DELAY	0.46
DEP_DEL15	1.00
DAY_OF_MONTH	0.00
DAY_OF_WEEK	0.00
CRS_ELAPSED_TIME	0.01
DISTANCE	0.01
is_prev_delayed	0.06
perc_delay	0.02
pagerank	-0.01
ELEVATION	-0.02
HourlyAltimeterSetting	-0.03
HourlyDewPointTemperature	0.03
HourlyWetBulbTemperature	0.04
HourlyDryBulbTemperature	0.04
HourlyPrecipitation	-0.07
HourlyStationPressure	-0.00
HourlySeaLevelPressure	-0.03
HourlyRelativeHumidity	-0.01
HourlyVisibility	-0.02
HourlyWindSpeed	0.04

Spearman Correlation



**Worst Airlines by
Percentage of
Flights Delayed**

- SouthWest
- JetBlue
- Frontier

**Best Airlines by
Percentage of
Flights Delayed**

- Hawaiian
- Delta

Nash 40

Our joined data set was both sufficiently large and considerably clean, containing 41 million flight records with 96% of them completely free of missing values across all 32 selected features.

Of these flights, 35 Million were Not Delayed and 6 Million were Delayed, giving us about a 17% rate of delayed flights.

When examining each airline's performance regarding flight delays relative to their total flights, we found that SouthWest had the highest percentage delayed

Hawaiian Airlines had the lowest by a significant margin but had considerably fewer total flights than its competitors.

Additionally, we used a spearman correlation between our features and our target variable, and we found that hourly precipitation had the strongest association.

	<h1>Engineered Features</h1>	
	<div> <div> Flight Delay Tracker States whether the flight was recently delayed <i>(is_prev_delayed)</i> </div> <div> Flight Diverted Tracker States whether the flight was diverted <i>(is_prev_diverted)</i> </div> <div> Extreme Weather Indicators Indicates active severe weather conditions <i>(multiple features)</i> </div> <div> Special Day Tracker Indicates special days and flight volume impacts <i>(AssumedEffect_Text)</i> </div> <div> Airline Efficacy Score Reports how frequently airline delays flights <i>(perc_delay)</i> </div> <div> Outgoing Flight Frequency Charts Airport Volume using PageRank <i>(pagerank)</i> </div> </div>	
	7	

Nina (1 min 13 sec)

Next, we focused on feature engineering.

We first created 2 flight tracking features. Specifically we created:

- A flight delay tracker, which is an indicator feature that states whether an aircraft was delayed. We assume that an aircraft which was delayed will likely have its subsequent flights delayed as well. This feature turned out to be highly predictive
- Similar to the flight delay tracker, we created an indicator that tracks whether an aircraft was diverted to a different airport. To our surprise, this feature actually caused our model performance to drop and so we left it out from our model

Next, we created 2 features that describe the physical environment, which includes:

- A series of indicator features that identify extreme weather conditions such as freezing rain and blowing snow
- We also tracked if the 'scheduled flight date' is likely to have abnormal air traffic volume due to holidays or shelter in place restrictions from the pandemic.

Finally, we created 2 features that describe the status quo of the airline industry:

- We created a measure of airline efficiency by calculating the percentage of

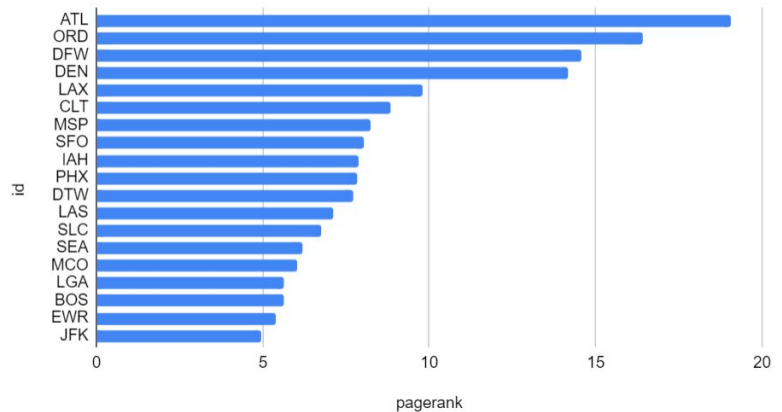
- flights that were delayed by that airline
- We also abstracted how important an airport is by calculating the page rank from their outgoing flights.

PageRank By Outgoing Flights

Top Airports by PageRank:

1. ATL - Atlanta International
2. ORD - O'Hare International
3. DFW - Dallas/Fort Worth International
4. DEN - Denver International
5. LAX - Los Angeles International

Pagerank By Outgoing Flights



Nash 17

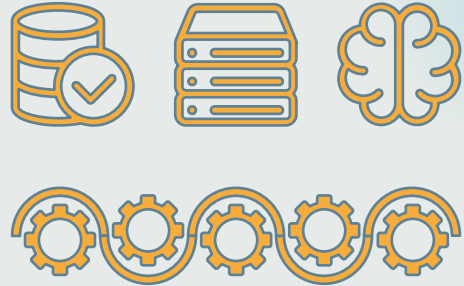
Calculating the pagerank by outgoing flight volume allowed us to see where airports ranked regarding their significance in outgoing flight volume and thus which are the busiest.

The airport with the highest score was ATL - Atlanta International Airport

- ORD - O'Hare International Airport
- DEN - Denver International Airport
- DFW - Dallas/Fort Worth International Airport
- LAX - Los Angeles International Airport

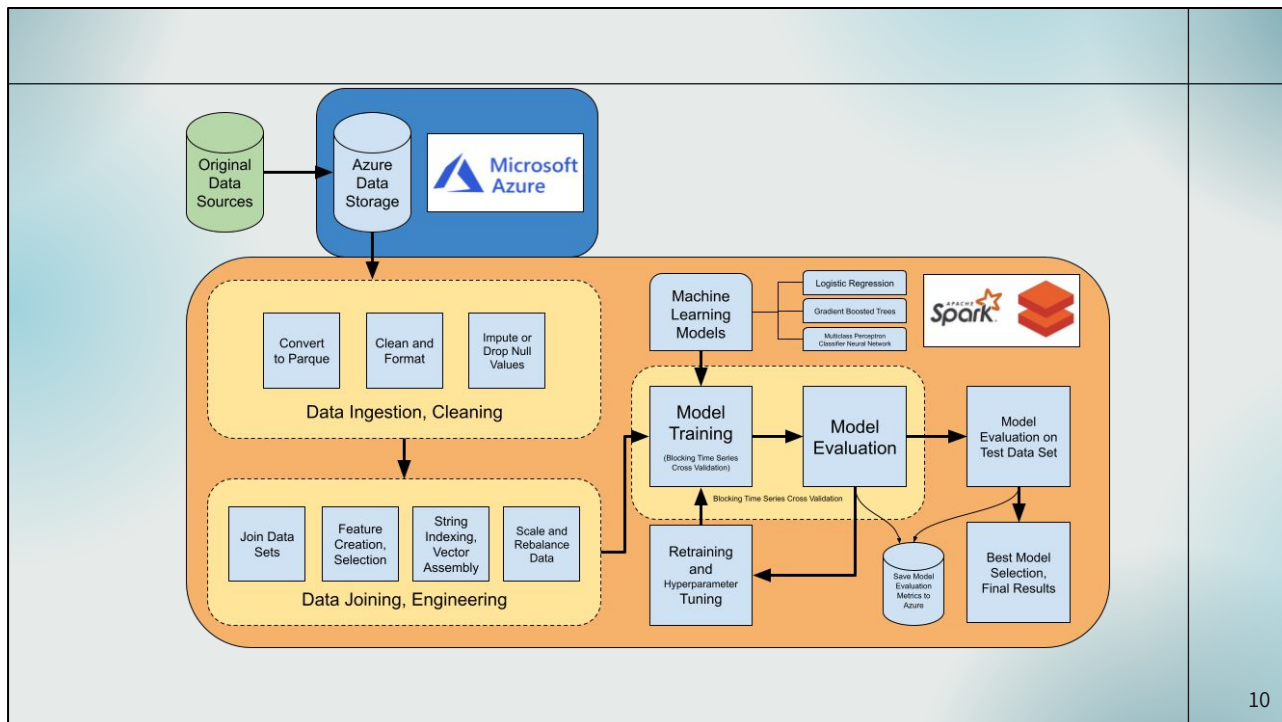
Data Pipeline

Data Ingesting and Machine Learning Modelling



Ryan ~ 17seconds

Now that we've gone over the data, let's talk about the data pipeline. This takes in the raw data, transforms it, and then uses it in machine learning.



Ryan - 18seconds

This is a diagram of our end-to-end data pipeline, which shows the steps involved in our project, including the data transformation tasks like string indexing, one-hot encoding, and vector assembling. We leveraged a number of tools and platforms such as Microsoft Azure, Apache Spark, and MLLib.

Special Pipeline Considerations I

- Model Integrity
 - 2015 dropped to prevent data leakage
 - Cancelled Flights == Delayed Flights
 - Time Lag and Timezone Conversion
 - Blocking Time Series Cross Validation

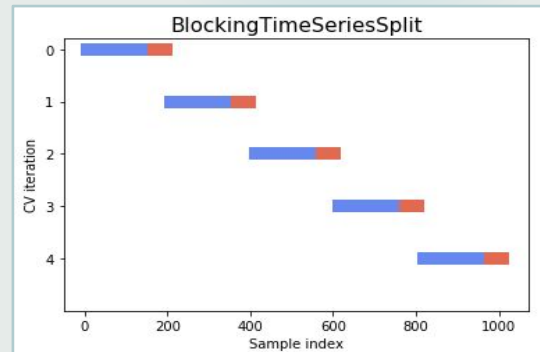


Image Credit:
<https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4>

11

Ryan - 11seconds

There were some special considerations that we had to make in our pipeline, thereby ensuring that our process is conceptually sound and mathematically correct.

We dropped flights from 2015 to prevent data leakage. Don't worry: it did not affect our metrics.

We treated cancelled flights as delayed flights.

We implemented a 2 hour time lag, and converted all the times to the same UTC time zone.

And we properly handled the time series data with a custom version of blocking time series cross validation,- pictured here.

Special Pipeline Considerations II

- Model Performance Optimization
 - Feature Selection
 - Sped up model training by dropping unimportant categorical features.
 - Rebalance Data with Downsampling
 - Improved model training performance by ensuring more equal distributions



Ryan - 19seconds

Furthermore, removing some unimportant, high dimensionality categorical features reduced training run-times from 24 hours to 1.5 hours without affecting our metrics.

And rebalancing our data with downsampling allowed our model to train better and make better predictions.

Models and Evaluation



13

Ryan ~ 6 seconds

With the pipeline explained, we can now move on to talking about the models and our evaluations.

Evaluation Metrics



False Negative

You waste some time.



False Positive

You miss your flight.

Primary Metric:	F0.5 Value
Secondary Metric:	Precision

Ryan - 37 seconds

When choosing evaluation metrics, remember our use case of informing users of flight delays: A false negative would mean you waste a little time. But a false positive would mean that you would miss your flight entirely. For this reason, we wanted to track a precision score to indicate how many worst-case scenarios we will have. But ultimately, we wanted our primary metric to balance prioritizing minimizing the false positives while still considering the false negatives. Thus, we chose F0.5 as our primary metric, which is a weighted average between precision and recall that penalizes false positives more so than false negatives.

Training, Test Data

- **Training Data:** Years 2016 - 2020
- **Test Data:** Years 2021

Models Created

- Logistic Regression, Baseline (Baseline LR)
- Logistic Regression, Feature Engineering (Feature Engineering LR)
- Multilayer Perceptron Classification Neural Network (MLP NN)
- Gradient Boosted Tree (GBT)
- Random Forest - Unfinished
- Linear Support Vector Machines - Unfinished

Cluster Information:

Databricks Clusters

1-10 Workers
16-160 GB Memory
4-40 Cores

Runtime11.3.x-cpu-ml-s
cala2.12

15

Ryan - 24 seconds

We chose to train the models on data from the years 2016 - 2020. And we tested the models on data from the years 2021.

We chose to pursue 6 models at first before ultimately reducing it down to four; the other two did not perform well enough to keep. Note that the basic baseline model was a logistic regression model without any of our newly engineered features.

Hyperparameter Tuning

Logistic Regression:

- *Regularization Parameter:* 0.0, 0.01, 0.5, 1.0, 2.0
- *Elastic Net:* 0.0, 0.5, 1.0
- *Maximum Iterations:* 5, 10, 50
- *Threshold:* 0.5, 0.6, 0.7, 0.8

Gradient Boosted Trees:

- *Maximum Iterations:* 5, 10, 50
- *Maximum Depth:* 4, 8, 16
- *Maximum Bins:* 32, 64, 128
- *Step Size:* 0.1, 0.5
- *Threshold:* 0.5, 0.6, 0.7, 0.8

Multilayer Perceptron Classifier Neural Network:

- *Maximum Iterations:* 100, 200
- *Block Size:* 128, 256
- *Step Size:* 0.03, 0.1
- *Threshold:* 0.5, 0.6, 0.7, 0.8
- *MLP NN Layer Architectures Used:*
 - [90, 30, 15, 2]
 - [90, 15, 2]

Ryan - 7 seconds

Lastly, each model was hyperparameter tuned with a number of parameters during training, all of which are listed here.

Logistic Regression - Baseline

Test Evaluation

regParam = 0.0 | elasticNetParam = 0.0 | maxIter = 5 | threshold = 0.5

0.20

F0.5

0.33

Precision

regParam = 0.0 | elasticNetParam = 0.0 | maxIter = 5 | threshold = 0.5

17

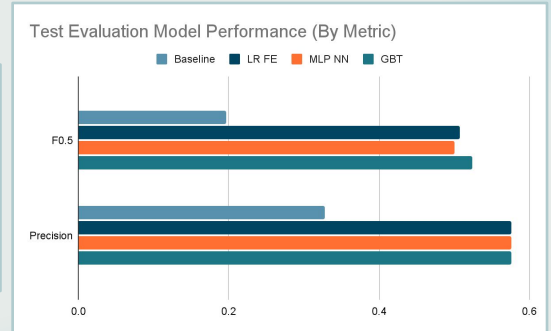
Deanna - 22s

We started with a baseline model, logistic regression, which was trained on data excluding the newly engineered features. When evaluated on our test data, the model performed poorly, with an F0.5 score of 0.2, and a precision of 0.3. Our goal was to improve upon this baseline with feature engineering and advanced machine learning.

Models Trained

- Logistic Regression (with engineered features) (LR FE)
- Multilayer Perceptron Neural Network (MP NN)
- Gradient Boosted Tree (GBT)
- Support Vector Machine
- Random Forest

Model	test_Precision	test_Recall	test_F0.5
Logistic Regression	0.575585	0.342514	0.506635
Gradient Boosted Tree	0.60977	0.335231	0.523952
Multilayer Perceptron	0.53993	0.386779	0.500309



Deanna - 12s

We trained and tuned a number of machine learning algorithms, including logistic regression with the engineered features added in, a multilayer perceptron neural network, gradient boosted trees, support vector machines, and random forest models.

Best Model!

Gradient Boosted Trees

maxIter = 5 | maxDepth = 4 | maxBins = 32 | stepSize = 0.5 | threshold = 0.6

Test Evaluation

0.52

F0.5

0.61

Precision

Improvement over Baseline

166%

F0.5

86%

Precision

27 features | 15 minute training runtime

19

Deanna - 13s

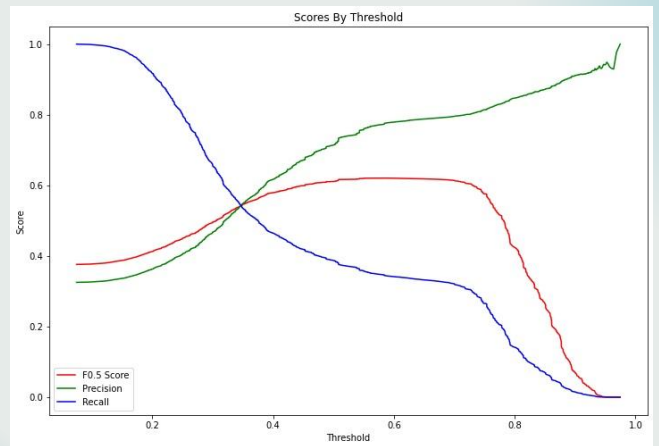
Of the models trained, Gradient Boosted Trees performed the best, with an F0.5 score of 0.52 and a precision of 0.61, which are 166% and 86% improvements from the baseline model respectively! Even with this significant lift, there is still room for improvement since we are seeing some evidence of overfitting.

Gradient Boosted Trees

maxIter = 5 | maxDepth = 4 | maxBins = 32 | stepSize = 0.5 | threshold = 0.6

Most Influential Features

- Previous flight delay indicator
- Departure hour
- Airline carrier
- Extreme weather indicator



Deanna - 13s

The model trained in about 15 minutes on 27 features. The top influential features for this model included the previous flight delay indicator, departure hour, airline carrier, and extreme weather indicators, most of which were among our engineered features.

Additional Experiments

- No-Weather Feature Set VS Feature Engineered Feature Set
- Only-Weather Feature Set VS Feature Engineered Feature Set
- Ensembling Methods and Performance Comparisons
- Changing Downsampling Ratios and Performance Comparisons

Deanna - 16s

Beyond these models, we conducted additional experiments, such as removing weather or flight related features before training, implementing models in conjunction using various ensembling methods, and changing the extent of our downsampling, but none helped to improve upon our best model.

Additional Info

Risks, Challenges, Limitations, and Next Steps



Nina 2 sec

So how did we do?

Gap Analysis

Joined Row Counts

~41 million rows

Metrics Used

F0.5 VS F1

Model Selection

GBT Very Common

Engineered Features

Flight delay tracker,
special weather indicators

Number of Features

27 VS 20-50 Features

Pipeline

Scaling, Dataset
Balancing

Nina (56 sec)

Our final model performance positions us well on the leaderboard where we had a similar F.5 score with other teams and above average on precision.

In addition to helping us identify where we stand among other teams, the leader board is also a critical source of inspiration for us to improve our model performance against our baseline result. By comparing our results to top performing teams with a post join dataset of at least 40 million records, we got several improvement ideas. Specifically we were inspired to update our metrics to F0.5 so that not only can we continue to focus on precision but also account for the general predictive power of our model. We were also inspired to create a GBT model, which turned out to be our top performing model.

Without going into the details, we were inspired to create 2 more engineered features among the total 27 features we selected for modeling. We also fine tuned our pipeline techniques.

Performance and Scaling Risks

Sorting for CV

Sorting by date can be expensive.

Custom CV Implementation

Jerry-rigged solution not optimal?

Dimensionality of Features

More data leads to major slowdown.

Data Streaming

Need to have the most current data!

Data Drift

Models require constant retraining.

24

Deanna - 44s

Despite our satisfaction with our model, there are some performance considerations to keep in mind when implementing this model into production.

1. First - sorting the data for cross validation is expensive, especially when we've created our own functions that may not be efficient.
2. Next - as we increase the number of features, the model slows down considerably, and for categorical features, if we encounter any unseen categories, they get dropped
3. Also - for the model to be of any value, data would need to be streamed such that it is current up to 2 hours behind real time, which could be costly to maintain.
4. Finally, the strengths of variable relationships shift over time, causing models to become outdated. Thus we would need to retrain models and track performance periodically.

Conclusion and Next Steps



Gradient Boosted Trees

F0.5: 0.52
Precision: 0.62

Future Work

Weighting by Year

Additional Features

Additional Models

Pipeline Improvements

Ryan - 41 seconds

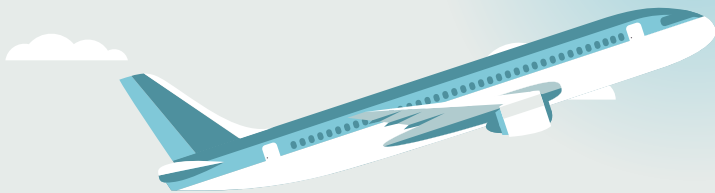

In conclusion, we are pleased with the significant improvement in the predictive power of our GBT model over the baseline model, which will allow our customers to better plan their travel time.

While our F0.5 and precision scores position us well on the leaderboard, there is always room for improvement.

With additional time, we would have liked to add a weighting scheme so that earlier years or outlier years are given less importance in model training.

There are also many more features to implement, such as destination weather, weather forecasting, other graph features, and binning of existing features.

And there are always other models to test, or changes to make to the pipeline that could help improve our performance.

	 <h2><i>Special Thanks!</i> </h2> <ul style="list-style-type: none">● Professor Vinicio De Sola: Instruction and Guidance● Mai La: Technical Assistance with GraphFrames● Max Eagle: Assistance with Run-Times, Feature Dimensionality
26	

Ryan - 5 Seconds

Shout outs and special thanks to the people listed here for their assistance with this project.

Project Credit Assignments

Task	Start Date	End Date	Estimated Time (hours)	Nashat Cabral	Deanna Emery	Nina Huang	Ryan S. Wong
Project Abstract Section	10/23/2022	10/30/2022	1	X		X	
Data Overview Section	10/23/2022	10/30/2022	4	X			
The Desired Outcomes and Metrics Section	10/23/2022	10/30/2022	2				X
Data Ingesting and Pipeline Section	10/23/2022	10/30/2022	4				X
Joining Datasets Section	10/23/2022	10/30/2022	4		X		
Machine Learning Algorithms to be Used Section	10/23/2022	10/30/2022	2		X		
Resource Management & Performance Optimization Section	10/23/2022	10/30/2022	4			X	
Train/Test Data Splits Section	10/23/2022	10/30/2022	2			X	
Conclusions and Next Steps Section	10/23/2022	10/30/2022	2	X		X	
Open Issues or Problems Section	10/23/2022	10/30/2022	2	X		X	
Set up Databricks instance	10/23/2022	10/30/2022	2				X
Set up GitHub and Integrate with Databricks	10/23/2022	10/30/2022	1				X
Phase 1 Submission	10/23/2022	10/30/2022	1				X
Revise Phase 1 Project Proposal	10/31/2022	11/13/2022	1	X	X		
Raw Data EDA: Airlines	10/31/2022	11/13/2022	2			X	X
Raw Data EDA: Stations and Weather	10/31/2022	11/13/2022	3	X			
Join Data Sets	10/31/2022	11/13/2022	6		X		
Initial Data Pipeline Creation	10/31/2022	11/13/2022	6				X
Model Implementation and Evaluation	10/31/2022	11/13/2022	3				X
Join Data Sets v2	10/31/2022	11/13/2022	6		X		
Implementing Time Series Cross Validation	10/31/2022	11/13/2022	3	X			X
Feature Engineering: Lag Time Window	10/31/2022	11/13/2022	3		X	X	
Feature Engineering: Time Zone Conversion	10/31/2022	11/13/2022	3			X	X
Feature Engineering: Airport Tracker	10/31/2022	11/13/2022	3				X
Feature Engineering: Flight Tracker	10/31/2022	11/13/2022	4				X
Feature Engineering: Previously Delayed	10/31/2022	11/13/2022	4				X
Data Cleaning: Imputation for Null Values	10/31/2022	11/13/2022	3		X		X
Phase 2 Check-In Presentation Video	10/31/2022	11/13/2022	2				X
Phase 2 Submission	10/31/2022	11/13/2022	1				X
Data Pipeline Creation	11/14/2022	11/25/2022	8			X	X
Model Building	11/18/2022	11/27/2022	12			X	X
Feature Engineering	11/14/2022	11/25/2022	10		X	X	X
Notebook Writeup	11/19/2022	11/27/2022	3		X	X	X
Presentation Setup	11/14/2022	11/16/2022	4		X	X	X
Phase 3 Submission	11/27/2022	11/27/2022	1			X	
Data Pipeline Revision	11/27/2022	12/4/2022	12			X	X
Model Building	11/27/2022	12/4/2022	20			X	X
Feature Engineering	11/27/2022	12/2/2022	12		X	X	X
Notebook Writeup	11/27/2022	12/4/2022	12		X	X	X
Presentation Setup	11/26/2022	12/4/2022	4		X	X	X
Gap Analysis	11/22/2022	12/4/2022	8				X
Phase 4 Submission	12/5/2022	12/5/2022	1		X	X	X
Final Presentation Preparation	12/5/2022	12/9/2022	12		X	X	X
Phase 5 Submission	12/9/2022	12/9/2022	1		X	X	X

Ryan

Here is our credit assignment table

Thank You!

Section 4, Group 1, Team 13

Sparks and Stripes Forever

Nashat Cabral: cabralnc96@berkeley.edu

Deanna Emery: deanna.emery@berkeley.edu

Nina Huang: ninahuang2002@berkeley.edu

Ryan S. Wong: ryanswong@berkeley.edu

Notebook Link

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon** and infographics & images by **Freepik**



Ryan - 5 Second

Thank you everyone!

Notebook Link:

<https://adb-731998097721284.4.azuredatabricks.net/login.html?o=731998097721284#notebook/632558266974488/command/1215577238246617>



Questions?

29

Ryan

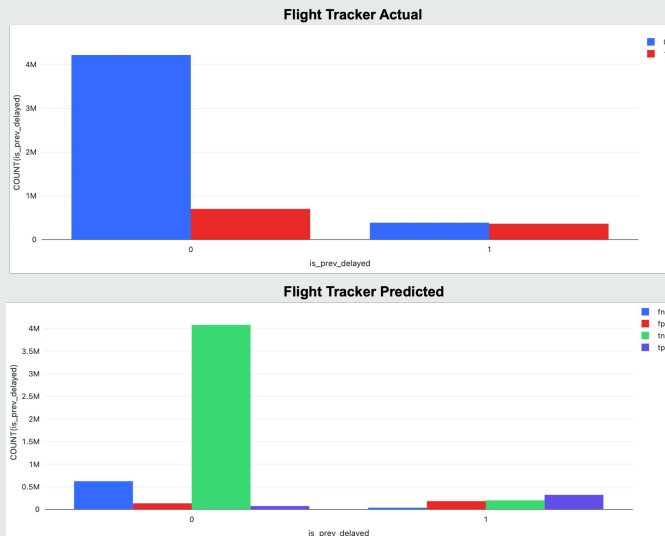
Are there any questions?

Appendix

- Interesting error analysis findings:
 - Delayed flight tracker
 - DEP Hours
 - Airline Effectiveness
 - Time drift: changes in correlation
- Code snippet:
 - Feature engineering via windowing: flight tracker (is_prev_delayed)
 - Pagerank calculation
 - Cross validation
- Interesting feature observations:
 - Airline efficiency
 - Time drift: changes in correlation
 - Extreme weather indicators

Appendix I: Interesting Error Analysis Findings

Error Analysis - delayed flight tracker



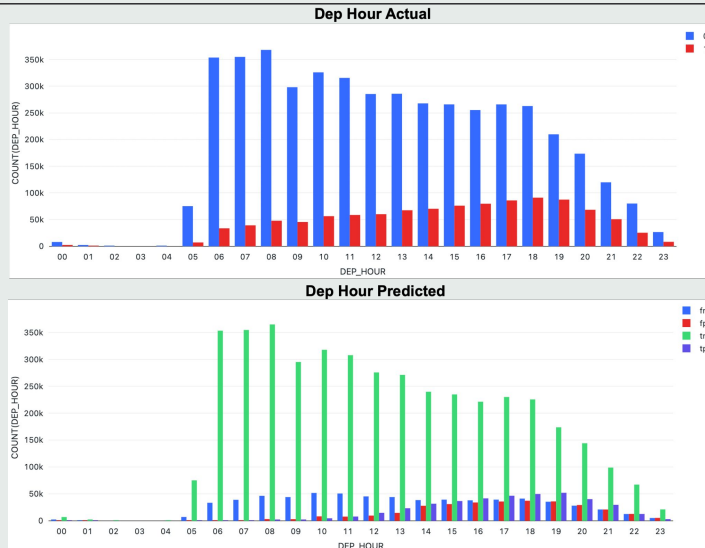
32

is_prev_delayed: This family of features is created to track whether the aircraft for a given flight is delayed due to a delay with the prior flight.

We observe that when the previous flight is delayed, that is a highly predictive signal that the next flight will also be delayed.

We can see that there is a higher proportion of false negatives when the previous flight wasn't delayed, which makes sense given our metrics priority. It is interesting that the proportion of true positives and false positives when the prior flight was not delayed is similar. This can be explained partly by our EDA analysis, such that the flight tracker is more predictive when the previous flight was delayed. When focusing solely on cases where the prior flight is delayed, we noticed only a small proportion of false negatives. As such, we believe that the `is_prev_delayed` variable behaves consistently between the holdout dataset and the training dataset.

Error Analysis - DEP Hours



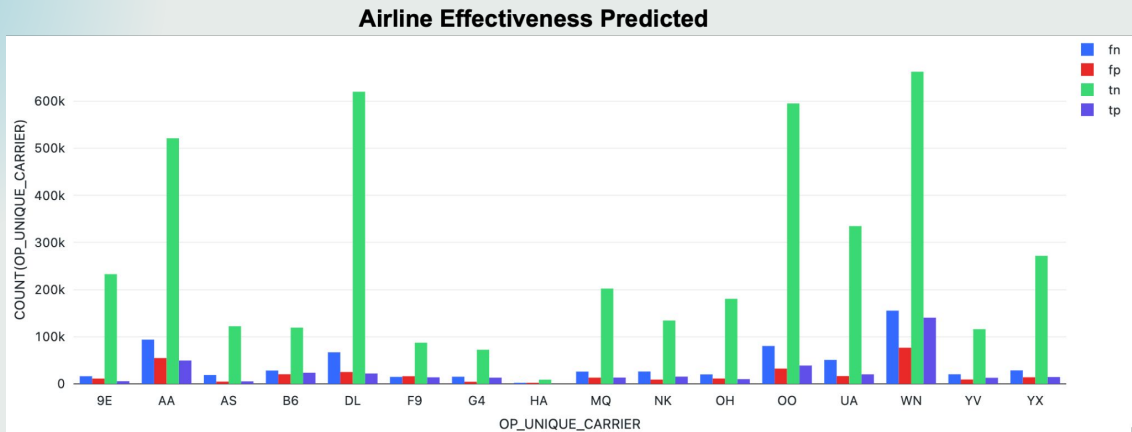
33

minimum delays are observed between 11PM – 5AM for both the predicted results (broken down into fn for false negative, fp for false positive, tn for true negative, and tp for true positive) and the actual results (calculated using DEP_DEL15).

Between 6AM-11AM we observe moderate occurrences of flight delays, and we see that there are significantly more false negatives than there are false positives. During peak flight delay hours (2PM – 8PM), the proportion of false positives and false negatives are more balanced. We are not surprised by this result where we see a higher proportion of false negatives compared to false positives given our priority to maximize F0.5 and precision.

That said, the proportion of false negatives to false positives changes during the day, which suggests certain hours of the days are more influential to the model than others and that other features might have a stronger pull on the model performance during less predictive hours.

Error Analysis - Airline Effectiveness



34

OP_UNIQUE_CARRIER: as noted previously from our correlation analysis, we noticed that the percentage effectiveness of airlines (percentage of flights delayed) against the outcome variable changes across the year. Specifically, we saw that the correlation strength dropped in 2020 and 2021 as a part of recovery from the major industry disruptor (Covid). As such, we were not surprised to see that although the effectiveness of the airline still played an important role in the modeling, the proportion of false positives was higher than true positives across big airlines such as AA and DL (see figure below). This confirms our correlation trend analysis, suggesting that our model performance could improve by further analyzing airline effectiveness across years for airlines of different sizes.

Time Drift: Changes in Correlation

Spearman Correlation

all 2016-2019 2020 2021

	DEP_DEL15	DEP_DEL15	DEP_DEL15	DEP_DEL15
YEAR	-0.01	0.01	nan	nan
DEP_DELAY	0.46	0.50	0.55	0.67
DEP_DEL15	1.00	1.00	1.00	1.00
DAY_OF_MONTH	0.00	-0.00	0.01	0.02
DAY_OF_WEEK	0.00	-0.00	0.01	0.02
CRS_ELAPSED_TIME	0.01	0.01	0.01	0.03
DISTANCE	0.01	0.01	0.00	0.04
is_prev_delayed	0.06	0.14	0.29	0.30
perc_delay	0.02	0.05	0.01	0.02
pagerank	-0.01	-0.01	-0.01	-0.03
ELEVATION	-0.02	-0.02	0.00	-0.01
HourlyAltimeterSetting	-0.03	-0.03	-0.01	-0.06
HourlyDewPointTemperature	0.03	0.03	-0.04	0.08
HourlyWetBulbTemperature	0.04	0.04	-0.05	0.11
HourlyDryBulbTemperature	0.04	0.04	-0.06	0.11
HourlyPrecipitation	-0.07	0.03	0.06	0.07
HourlyStationPressure	-0.00	0.00	-0.01	-0.02
HourlySeaLevelPressure	-0.03	-0.03	-0.00	-0.06
HourlyRelativeHumidity	-0.01	-0.01	0.02	-0.03
HourlyVisibility	-0.02	-0.02	-0.04	-0.02
HourlyWindSpeed	0.04	0.05	0.04	0.03

Other interesting fun observations from correlation analysis:

1. We focused on Spearman correlation to measure the strength and direction of association (monotonic association)
2. Engineered features have a higher Pearson correlation than they do with Spearman. This suggests that while our engineered features can improve our logistic regression model results (assumes linearity), they may not be able to noticeably lift tree-based models results given the non-monotonic nature of the features

Appendix II: Code Snippet

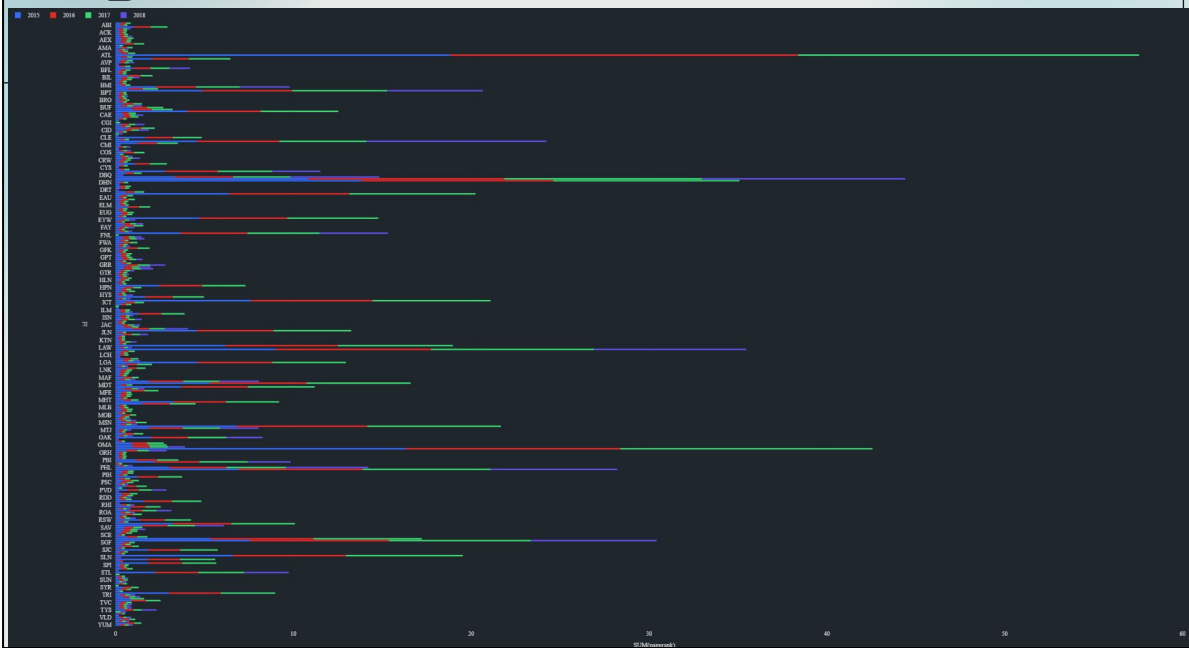
Flight Tracker: is_prev_delayed

```
1 def get_flight_tracker(df):
2     # prep the dataframe lag calculation
3     df = df.sort(df.OP_UNIQUE_CARRIER.asc(), df.FL_DATE.asc(), df.TAIL_NUM.asc(), df.DEP_TIME.asc())
4
5     # Helper Features
6     # Create partition key
7     w = Window.partitionBy("FL_DATE", "TAIL_NUM").orderBy("DEP_TIME_CLEANED")
8
9     # create columns over the partition key such that we obtain delays from the prior flight
10    # also grab the 2 flight prior in case the difference between an aircraft's flights is within 2 hrs
11    is_1prev_delayed = F.lag("DEP_DEL15", 1).over(w)
12    is_2prev_delayed = F.lag("DEP_DEL15", 2).over(w)
13    df = df.withColumn("is_1prev_delayed", is_1prev_delayed)
14    df = df.withColumn("is_2prev_delayed", is_2prev_delayed)
15
16    # create temp helper data to see if there is a need to run 2 flights prior to avoid data leakage
17    prev_dep_time = F.lag("DEP_TIME_CLEANED", 1).over(w)
18    df = df.withColumn("prev_dep_time", prev_dep_time)
19    df = df.withColumn("dep_time_diff", (df["DEP_TIME_CLEANED"] - df["prev_dep_time"]))
20
21    # all helper features will not be selected for modeling
22    # now create our flight tracker column that will be used for modeling
23    df = df.withColumn(
24        "is_prev_delayed",
25        F.when((df["dep_time_diff"]) >= 200, df['is_1prev_delayed'])
26        .otherwise(df['is_2prev_delayed']))
27
28    # replace null with 0, meaning first flight of the date (with null) is not delayed
29    df = df.na.fill(value = 0, subset=['is_prev_delayed'])
30
31    return df
```

Pagerank Calculation

```
1  from graphframes import *
2  #Non-21
3  #2015
4
5  df_joined_data_all_2015 = df_joined_data_all_N21.filter(df_joined_data_all.YEAR == "2015")
6
7  vertices = df_joined_data_all_2015.select('DEST','YEAR').withColumnRenamed('DEST','id').distinct()
8
9  edges = df_joined_data_all_2015.select('DEST','ORIGIN').withColumnRenamed("DEST","src").withColumnRenamed("ORIGIN","dst")
10
11  g = GraphFrame(vertices, edges)
12
13  results2015 = g.pageRank(resetProbability=0.15, tol=0.01)
14  display(results2015.vertices)
```

Pagerank



Custom Blocking Time Series Cross Validation

```
def runBlockingTimeSeriesCrossValidation(preppedTrain, featureCol='features', cv_folds=4, regParam_input=0, elasticNetParam_input=0,
                                         maxIter_input=10, thresholds_list = [0.5]):
    """
    """

    for i in range(cv_folds):

        print(f"Running fold {i+1} of {cv_folds}")
        print(f"@ {getCurrentDateTimeFormatted()}")
        min_perc = 1-cutoff
        max_perc = min_perc + cutoff
        train_cutoff = min_perc + (0.7 * cutoff)

        cv_train = preppedTrain.filter((col("DEP_DATETIME_LAG_percent") >= min_perc) & (col("DEP_DATETIME_LAG_percent") < train_cutoff))\
            .select(["DEP_DEL15", "YEAR", "DEP_DATETIME_LAG_percent", featureCol]).cache()

        cv_val = preppedTrain.filter((col("DEP_DATETIME_LAG_percent") >= train_cutoff) & (col("DEP_DATETIME_LAG_percent") < max_perc))\
            .select(["DEP_DEL15", "YEAR", "DEP_DATETIME_LAG_percent", featureCol]).cache()

        lr = LogisticRegression(labelCol="DEP_DEL15", featuresCol=featureCol, regParam = regParam_input, elasticNetParam = elasticNetParam_input,
                                maxIter = maxIter_input, threshold = 0.5, standardization = True)

        lrModel = lr.fit(cv_train)

        currentYearPredictions = lrModel.transform(cv_val).withColumn("predicted_probability", extract_prob_udf(col("probability"))).cache()

        print(f"Starting threshold search")
        for threshold in thresholds_list:
            print(f"Testing threshold {threshold}")

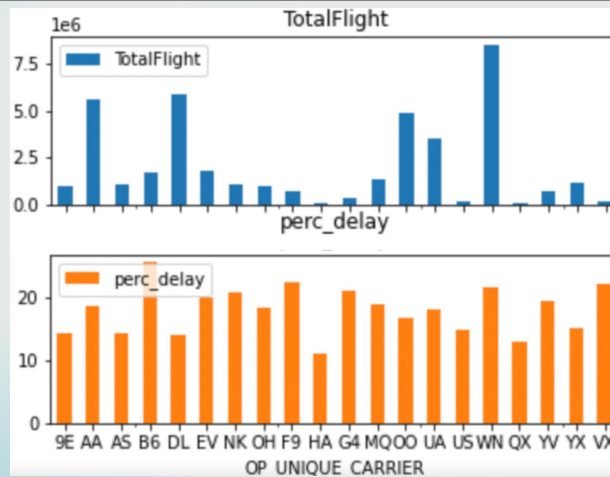
            thresholdPredictions = currentYearPredictions.select("DEP_DEL15", "predicted_probability")\
                .withColumn("prediction", (col("predicted_probability") > threshold).cast("double")).cache()

            currentYearMetrics = testModelPerformance(thresholdPredictions)
            stats = pd.DataFrame([currentYearMetrics], columns=['val_Precision', 'val_Recall', 'val_F0.5', 'val_F1', 'val_Accuracy'])
            stats['cv_fold'] = i
            stats['regParam'] = regParam_input
            stats['elasticNetParam'] = elasticNetParam_input
            stats['maxIter'] = maxIter_input
            stats['threshold'] = threshold
            stats['trained_model'] = lrModel

        cv_stats = pd.concat([cv_stats, stats], axis=0)
```

Appendix III: Interesting Feature Observations

Airline Efficacy Score

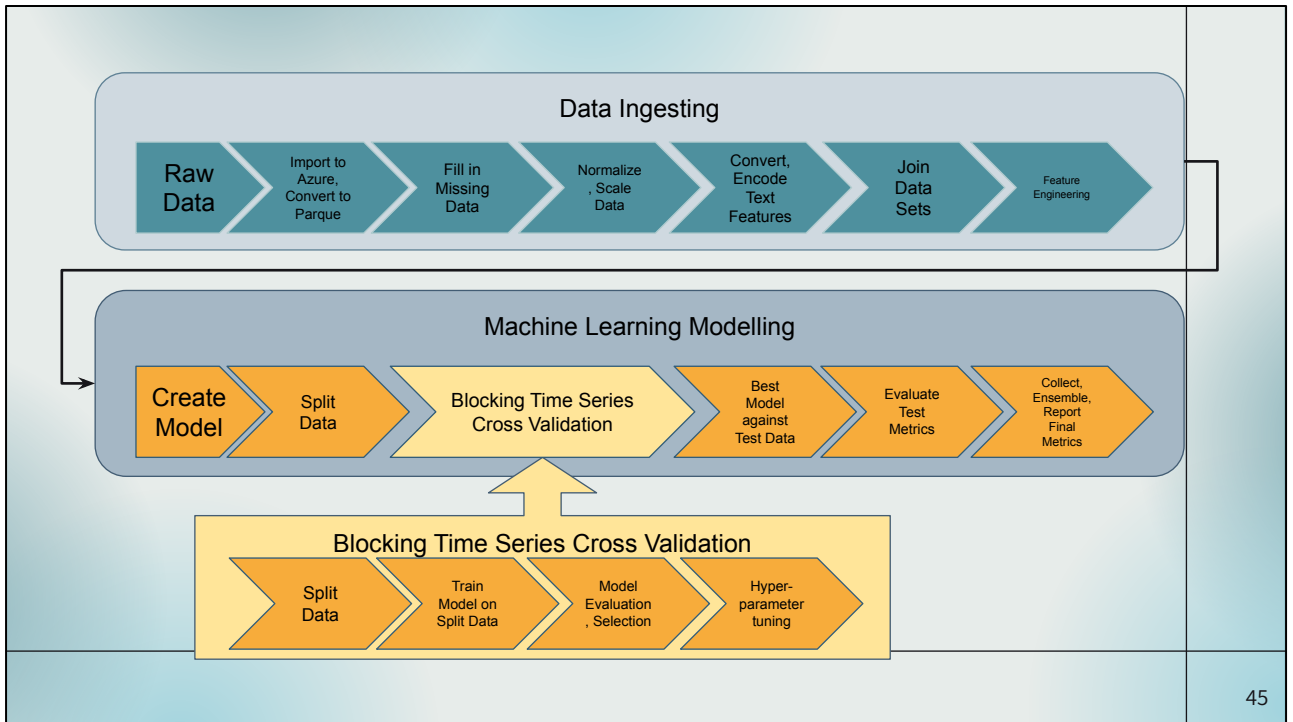


Total flight volume scheduled for an airline and their corresponding percentage delay

Extreme Weather Indicators

Weather Type	Count	% of Total Rows
Blowing_Snow	55930	0.2%
Freezing_Rain	101508	0.3%
Rain	6590540	19.0%
Snow	2475951	7.0%
Thunder	614697	1.7%

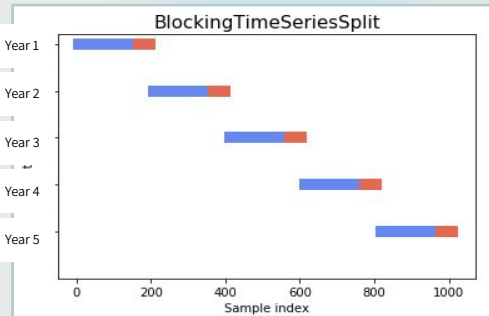
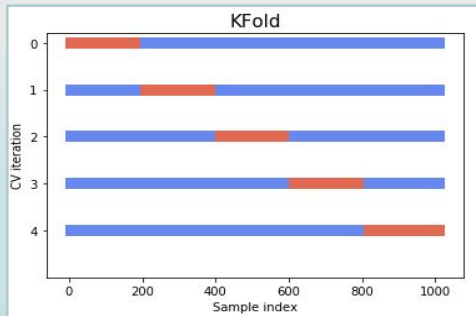
Weather Type	Count	% of Total Delayed Rows
Blowing_Snow	12337	0.2%
Freezing_Rain	25869	0.4%
Rain	1439607	22.3%
Snow	514283	8.0%
Thunder	286287	4.4%



Ryan ~ 35 seconds

The first half of the pipeline focuses on ingesting and refining the data. From raw data, we save it to Azure in the Parquet format, making it highly available in a computationally efficient format. For missing data, we impute values that can be reasonably inferred, and drop rows where the missing data is ambiguous and duplicate rows. We then convert text features with StringIndexing and one hot encode them using sparse vectors, including those features in our analysis in a space efficient manner. Then, we create new features and join the data set to finish refining the data.

Blocking Time Series Cross Validation



Images From: <https://hub.packtpub.com/cross-validation-strategies-for-time-series-forecasting-tutorial/>

46

Ryan ~ 1 minute

Since we are evaluating time series data, we need to use time series cross validation for accurately training and evaluating our models. We could not use the standard KFold cross validation because it is flawed with time series: KFold introduces time gaps in the data, tests on data occurring before the training data, and it leaks data when the model memorizes future data it should not have seen yet. As such, we chose to build our own version of cross validation called BlockingTimeSeriesSplit. Blocking Time Series Split will split the training data by year, builds a model for each year, trains that model on data from the first 70% of that year, and then tests that model on the data from the latter 30%. The model with the best metrics when tested is chosen as our best model to be evaluated against the 2021 test data. This allows us to cross validate our time series data without the complications of KFold.

Logistic Regression - Feature Engineering

Test Evaluation

0.507

F0.5

0.576

Precision

Improvement over Baseline

157%

F0.5

75%

Precision

regParam = 0.0 | elasticNetParam = 0.0 | maxIter = 5 | threshold = 0.6

47

Ryan

So, about that Logistic Regression model. After being trained, cross-validated, and evaluated against our test data, this baseline model with the default parameters has an astounding precision of... 2.92%. F1 and recall were... slightly better? But such low precision was surprising.

Multiclass Perceptron Classifier Neural Network

Test Evaluation

0.500

F0.5

0.576

Precision

Improvement over Baseline

154%

F0.5

65%

Precision

maxIter = 100 | blockSize = 128 | stepSize = 0.5 | threshold = 0.5 | layers=[90,30,15,2]

48

Ryan

So, about that Logistic Regression model. After being trained, cross-validated, and evaluated against our test data, this baseline model with the default parameters has an astounding precision of... 2.92%. F1 and recall were... slightly better? But such low precision was surprising.

Gradient Boosted Trees

Test Evaluation

0.524

F0.5

0.576

Precision

Improvement over Baseline

166%

F0.5

86%

Precision

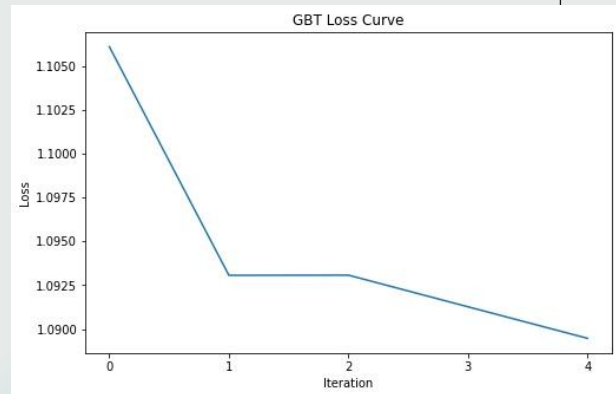
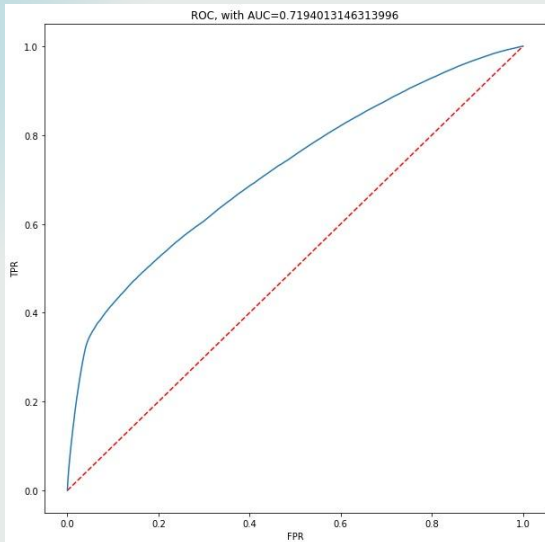
maxIter = 100 | blockSize = 128 | stepSize = 0.5 | threshold = 0.5

49

Ryan

So, about that Logistic Regression model. After being trained, cross-validated, and evaluated against our test data, this baseline model with the default parameters has an astounding precision of... 2.92%. F1 and recall were... slightly better? But such low precision was surprising.

GBT ROC and Loss Curve



Experiment: No-Weather Feature Set

- Logistic Regression - No-Weather Feature Set
 - Logistic Regression Model trained on feature set without weather features.
 - How will model predictive power change?

- Result:
 - Minor decrease in training validation F0.5, Recall
 - Negligible increase in everything else

Metric	Feature Engineered LR	No-Weather LR	% Improvement over FE
val_f0.5	0.639	0.631	-1.25%
val_precision	0.798	0.800	0.25%
val_recall	0.356	0.342	-3.93%
test_f0.5	0.513	0.516	0.58%
test_precision	0.625	0.626	0.16%
test_recall	0.298	0.303	1.68%

Experiment: Only-Weather Feature Set

- Logistic Regression - Only-Weather Feature Set
 - Logistic Regression Model trained on feature set with only weather features.
 - How will model predictive power change?

Metric	Feature Engineered LR	Only-Weather LR	% Improvement over FE
val_f0.5	0.639	0.256	-59.94%
val_precision	0.798	0.755	-5.39%
val_recall	0.356	0.070	-80.34%
test_f0.5	0.513	0.170	-66.86%
test_precision	0.625	0.483	-22.72%
test_recall	0.298	0.047	-84.23%

- Result:
 - Minor to catastrophic decrease in every metric

Experiment: Ensembling Methods

- Implement different ensembling methods on best LR model.
 - Ensemble Majority, Ensemble At Least One, and Ensemble Unanimous.
 - How will model predictive power change?

Model	test_Precision	test_Recall	test_F0.5	test_F1	test_Accuracy
Logistic Regression	0.576	0.343	0.507	0.429	0.829
Gradient Boosted Tree	0.610	0.335	0.524	0.433	0.835
Multilayer Perceptron	0.540	0.387	0.500	0.451	0.823
Ensemble Majority	0.607	0.338	0.524	0.434	0.835
Ensemble At Least One	0.510	0.405	0.485	0.452	0.815
Ensemble Unanimous	0.631	0.321	0.529	0.426	0.837

- Result:
 - Unanimous Ensemble achieved slightly higher F0.5, precision over GBT.

Experiment: Downsampling Ratios

- Implement different downsampling ratios on best LR model.
 - Standard Ratio: 1 Delayed - 1.5 Non-Delayed.
 - Experimental Ratio: 1 Delayed - 2 Non-Delayed
 - How will model predictive power change?

- Result:
 - Experimental Downsampling Ratio has worse F0.5, Precision

Model	Standard Downsample LR	Experimental Downsample LR	% Improvement Experimental over Standard
test_Precision	0.576	0.504	-12.5%
test_Recall	0.343	0.3742225	9.3%
test_F0.5	0.507	0.46951016	-7.3%

Interesting Problems Experienced

Messy Data

Bad Data Docs

2021 Year Glitch

**DataBricks Broke
GraphFrames**

**Not Enough
Time**

55

Deanna - 1 min

Throughout our analysis, we've come across some strange and interesting errors!

As mentioned before, the raw data was messy - notably, the data documentation was incomplete and at times inconsistent.

We also came across issues early on in the join phase with airport codes in the stations dataset that didn't quite always contain Ks at the beginning of the code. We also realized, having discovered that 2021 was entirely missing from our joined dataset, that the dates and times in the airports data changed formats in 2021.

Beyond the data, we have been facing issues with the DataBricks infrastructure that prevent us from using GraphFrames on more than a single node cluster.

Unfortunately, attempting to run something like PageRank on a single node would not be feasible given time constraints, so we have been exploring alternative methods.

This leads me to perhaps the largest challenge we face: it seems like we never have enough time to do everything we want to do!

Project Challenges

Learning Pipelines

New Tools, Platforms

Messy Data, Bad Docs

Custom Time Series CV

**SMOTE Nonfunctional,
GraphFrames,**

Data Dimensionality

56

(30 sec)

To overcome these problems, we explored with many approaches. First, we overcame the tech stack barrier by quickly ramping up our knowledge on how to get machine learning pipelines to work on Spark dataframes. We also resorted to creating user defined functions to clean messy data, perform time series CV, and balancing our data. To get the pagerank to work, we established a cluster that is compatible with GraphFrames. We also treated our one hot encoding with care by considering data dimensionality.

Project Limitations

Time Constraints

Not enough time to do everything.

MLLib Missing Functionality

Why can't it do ____?

Data Quality Concerns

Is all the data really true?

Limited Compute Resources

Compute resources
fickle, unreliable

57

Having reviewed challenges that we were able to overcome, we'd like to highlight some other limitations.

- First, It would be even better if we had time to try and explore more improvement techniques such as creating more user defined functions to compliment missing features from MLLib
- We also have doubts with the truthfulness of data, especially when we observed wind speed that are in the 3 digits
- Finally, we'd wish there was more computing resources so that we can spend more time on development as opposed to waiting for the model to run, or even worse, losing 11 hours of model results due to environment issues and having to rerun them

