W261 Final Presentation: Flight Delay Forecasting

Summer Hao, Mary Chau, Austin Jin, Matt Kinkley
Team 10

Agenda

- Introduce the Business Case
- Dataset Introduction
- EDA & Feature Engineering
- Join Strategy
- Algorithm Summaries
- E2E Pipeline
- Evaluation Metrics
- Leaderboard Result & Gap Analysis
- Experiments
- Limitations, Challenges, & Future Work

Problem & Mission

Problem

With flight delays causing a myriad of issues, airlines are baring costs for the associated inconvenience, maintenance, and economic losses.

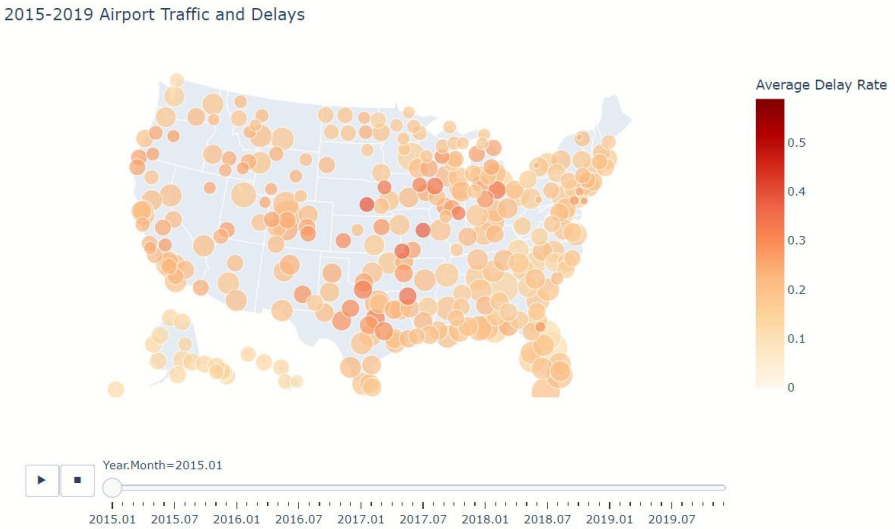
Mission

Team 10 consulting will enable all airlines to predict the potential of flight delays to help make them make the appropriate decisions through effective models to ultimately decrease costs and improve customer satisfaction.

Dataset Introduction

- **3 Data Files:** Airlines (31m+), Weather(600m+), Airport geolocation from OpenFlight
- **Sample:** 2015-2019
- **Frequency:** Airlines (minutes), Weather (minutes, measured hourly at each station)
- **Null Values:** filtering out variables with >50% missing rate, keep extreme weather variables with low breadth and fill na with 0.
- **Imputation Strategy:** median for numeric, 9999 for categorical, 0 for delay and traffic related variables, snow depth and precipitation depth.
- **Outliers**: log transformation.
- **Cyclical Features**: trigonometry transformation for seasonal features and wind direction.
- **Categorical Variables**: String Indexer and One Hot Encoding.
- **Output**: 31 million rows, 142 features belonging to seasonal, spatial, operational, weather groups.

EDA & Features



Filling Missing Data

- Numerical – Median
- Categorical - 9999

Transformation

log, one hot encoding, winsorization

Airport Traffic & Delay

The rolling 24HR total delay divided by rolling 24HR total traffic for origin, destination, and airline hubs.

Previous Aircraft Delay

The aircraft's most recent delay in minutes in last 24HR until 2HR prior to the current departure time

Expected Preparation Time

The time interval between the scheduled arrival time of the aircraft's most recent trip and the scheduled departure of the current trip.

Main Join Snippet

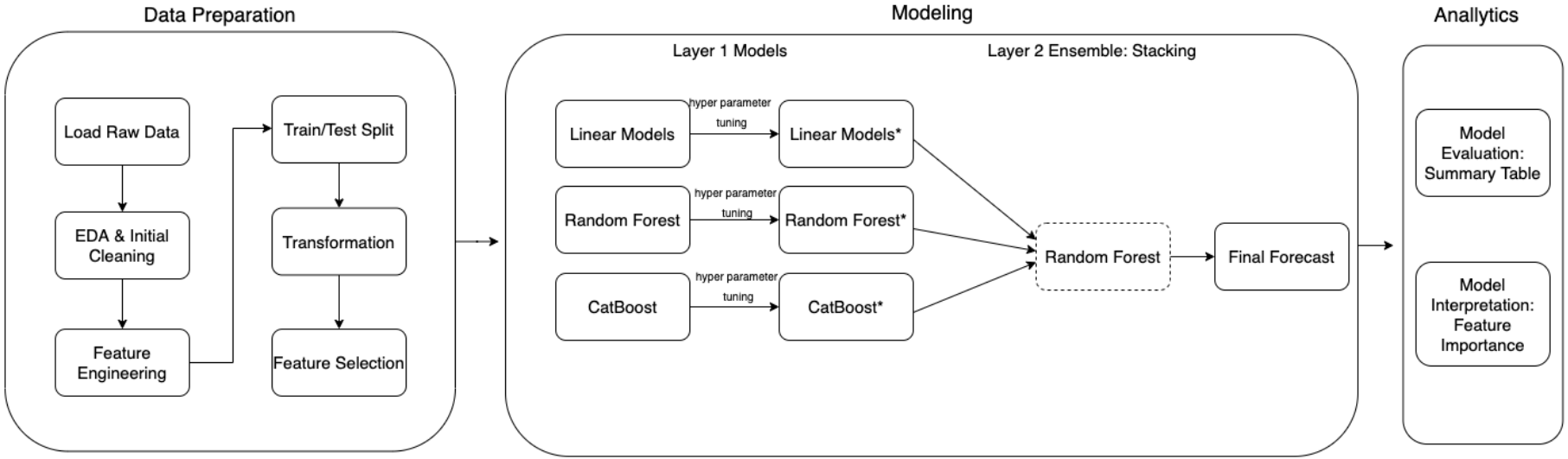
```
#create hourly buckets for weather data
df_weather_bkt = df_weather.withColumn("bucket_weather", f.date_trunc("hour","DATE"))
w = Window.partitionBy("TIMEr","FL_NUM").orderBy("dts_diff")
df_airlines_bkt = df_airlines.withColumn("bucket_airlines", f.date_trunc("hour","TIMEr"))\
                .union(df_airlines.withColumn("TIMEr_lag1", f.col("TIMEr")- f.expr("INTERVAL 1 HOUR"))\
                      .withColumn("bucket_airlines", f.date_trunc("hour","TIMEr_lag1")).drop("TIMEr_lag1"))\
                .union(df_airlines.withColumn("TIMEr_lag2", f.col("TIMEr")- f.expr("INTERVAL 2 HOURS"))\
                      .withColumn("bucket_airlines", f.date_trunc("hour","TIMEr_lag2")).drop('TIMEr_lag2'))

#join by buckets and filter for the closest matching weather record.
cond_join = (df_airlines_bkt.ORIGIN_AIRPORT_ID==df_weather_bkt.airport_id) & \
(df_airlines_bkt.bucket_airlines==df_weather_bkt.bucket_weather)
df_joined = df_airlines_bkt.join(df_weather_bkt, cond_join, how='left')\
            .withColumn("DATE", f.when(f.col("DATE").isNull(),f.col("TIMEr")).otherwise(f.col("DATE")))\
            .where("DATE<=TIMEr")\
            .withColumn("dts_diff", f.when(f.col("DATE").isNull(), 0)\
                        .otherwise(f.col("TIMEr").cast('long')-f.col("DATE").cast('long')))\
            .withColumn("row_num", f.row_number().over(w)).where("row_num==1")\
```

Algorithms Considered

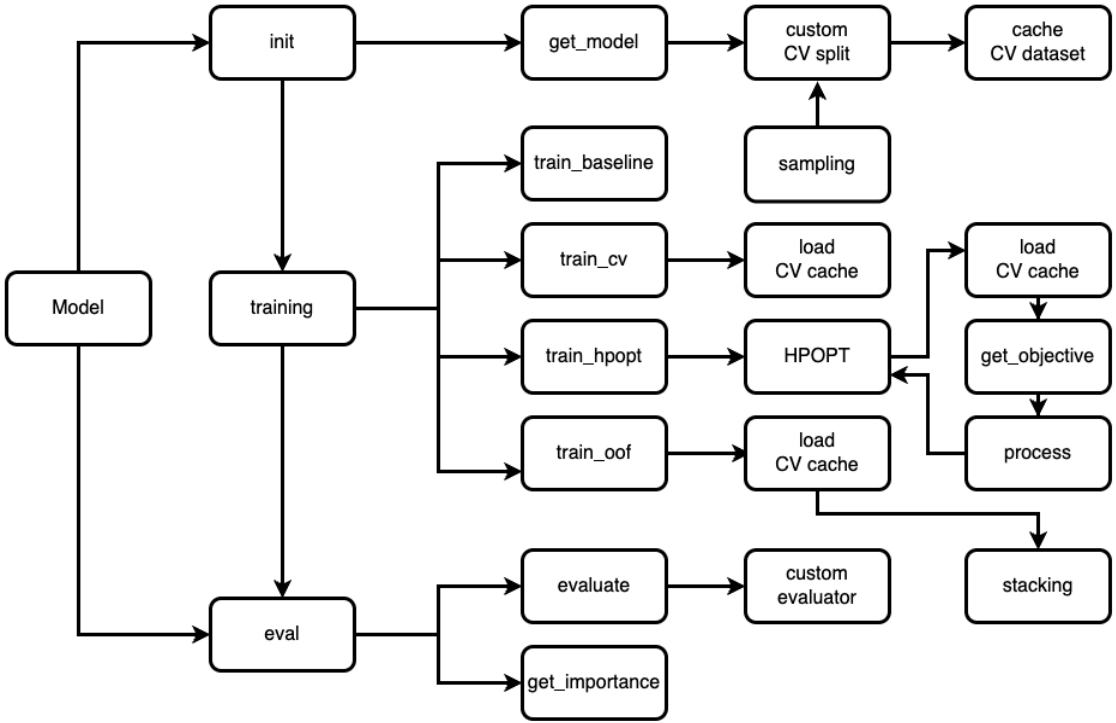
We considered/experimented with these classic ML algorithms as candidates:

Algorithm	Training Time	Performance	Spark Compatibility	Selected for Final	Challenge
Logistic Regression	**Fast**	Poor	**High**	**YES**	
Random Forest	Slow	Medium	**High**	**YES**	
Gradient Boost Tree	Slow	Medium	**High**	No	Slow Tuning
LightGBM	**Fast**	**Good**	Low	No	Memory Issue on big data
CatBoost	Slow	**Good**	Low	**YES**	▪ Spark cluster configuration ▪ Package installation conflict with LightGBM

Final End-to-End Pipeline



Model Pipeline Structure



Model Pipeline Snippet

```
# Hyperparameter grid
logistic_clf_param = {
    "regParam":        hp.quniform(0,2,0.1),
    "elasticNetParam": hp.quniform(0,1,0.01),
    "threshold":       hp.choice('threshold', [ 0.3, 0.4, 0.5, 0.6])
}

# Model Initialization
para = {}
para['model'] = 'linear_clf'     # model type
para['col_y'] = 'DEP_DEL15'      # label
para['imbalance_adj'] ='undersampling' # Sampling type. Options: [None, 'undersampling', 'oversampling']
para['multiplier'] = 0.5         # Sampling rate with multiplier value. Options: [0.5, 0.75, 1]
para['cv'] = 'ts'                # Cross-validation type. Options: ['ts', 'kfold']
para['useAvro'] = True
model_pl = model_pipeline(df_train, df_test, df_meta, para, ver=f"{dataset_ver}_scaled", load_data=True, save_data=False,
data_ver=dataset_ver)

# Train with Hyperparameter Search
best_model, model_trials = model_pl.train_hpopt(logistic_clf_param, tpe.suggest, 10, early_stopping_rounds=5)

#Eval best model
training_summary = model_pl.evaluate(best_model, model_trials)
training_summary_pivot = training_summary.reset_index().set_index(["index","stage"])

# Display Summary
training_summary_pivot

# Plot Feature Importance
importances = model_pl.get_importance(best_model)
df_plt = importances.loc[importances.importance>0.00001,:]
ax = df_plt.set_index('name').sort_values('importance').plot(kind='barh',figsize=(10,10))
ax.set_title('Feature Importance Best Logistic Regression')
```

Evaluation Metrics

Metric Selection Considerations

- Accommodate heavily imbalanced data
- Balance between Type I and Type II errors.

1. F1 score
2. Matthews Correlation Coefficient (MCC)
2. Cohen's Kappa Coefficient
3. Precision & Recall

Model Performance

	stage	Logistic Baseline	RF	CatBoost
fscore	Train	0.51	0.62	0.65
MCC	Train	0.39	0.56	0.59
cohen_kappa	Train	0.37	0.49	0.55
precision	Train	0.41	0.45	0.51
recall	Train	0.67	0.96	0.91
training_time	Train	3.23	16.17	26.74
fscore	Validation	0.50	0.61	0.64
MCC	Validation	0.39	0.56	0.58
cohen_kappa	Validation	0.38	0.49	0.55
precision	Validation	0.41	0.44	0.51
recall	Validation	0.63	0.96	0.88
training_time	Validation	2.36	28.99	62.61
fscore	Test	0.51	0.64	0.68
MCC	Test	0.38	0.58	0.61
cohen_kappa	Test	0.36	0.52	0.60
precision	Test	0.41	0.48	0.62
recall	Test	0.67	0.96	0.76

Leaderboard Results

Join: 7 min on full data

Cross Validation:

- logistic Regression: 3-5 min
- Random Forest: 30-40 min
- CatBoost: ~60 min

Number of Rows

- Training (2015-2018): 23.9M
- Test (2019): 7.2M

Number of Features

- Created & Considered: 142
- Selected: 53

Cross-Validation:

- F1 Score: 0.64

Blind Test (2019) Data:

- F1 Score: 0.68
- MCC: 0.61
- Cohen's kappa: 0.60

Top 5 Features

- PREV_DEP_DEL_MINUTES
- expect_prep_time
- hour_cosine
- total_delay_rate_origin
- OP_CARRIER_WN

Experiments Part I

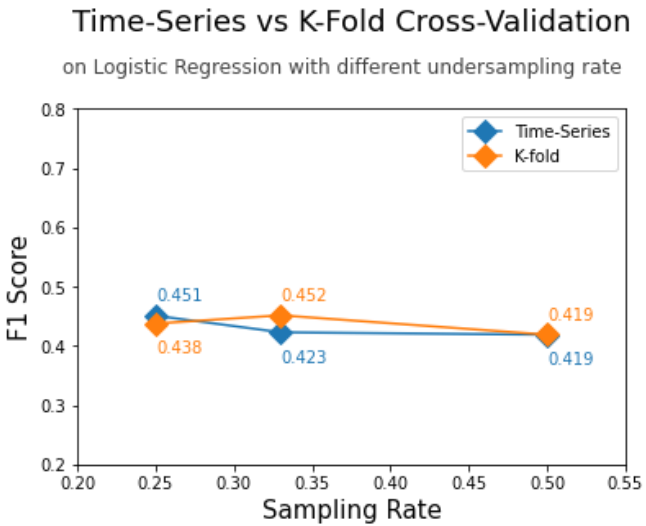
Big Data vs Small Data

No significant information lost from 142 to 53 features (-89)

	stage	RF Small Data (53)	RF Big Data (142)
fscore	Train	0.62	0.63
MCC	Train	0.56	0.56
cohen_kappa	Train	0.49	0.51
precision	Train	0.45	0.47
recall	Train	0.96	0.93
training_time	Train	16.17	33.20
fscore	Validation	0.61	0.61
MCC	Validation	0.56	0.56
cohen_kappa	Validation	0.49	0.51
precision	Validation	0.44	0.47
recall	Validation	0.96	0.90
training_time	Validation	28.99	42.20
fscore	Test	0.64	0.65
MCC	Test	0.58	0.58
cohen_kappa	Test	0.52	0.53
precision	Test	0.48	0.49
recall	Test	0.96	0.93
eval time	Test	1.82	2.38

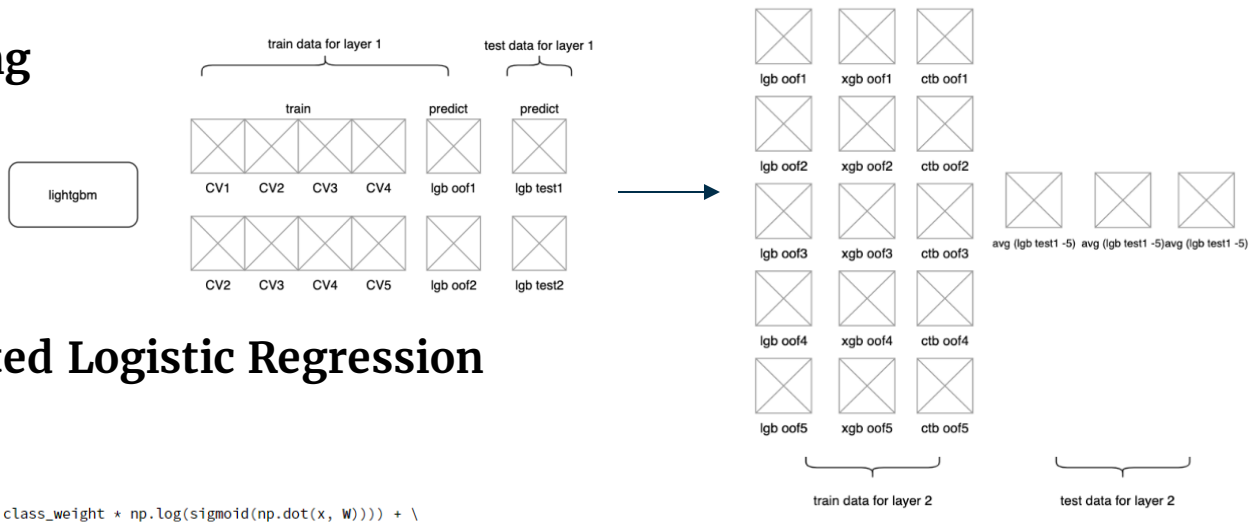
Time-Series vs K-Fold Cross-Validation

Similar F1 scores between blocked time-series and randomized k-fold split



Experiments Part II

2-Layer Stacking



RDD implemented Logistic Regression

```
def sigmoid(z):
  return 1.0/(1+np.exp(-z))

def logLoss(dataRDD, W):
  loss = dataRDD.map(lambda x: ((y * class_weight * np.log(sigmoid(np.dot(x, W)))) + \
                          ((1 - y) * class_weight * np.log(1 - sigmoid(np.dot(x, W)))))).mean()
  loss = -loss
  return loss

def GDUpdate(dataRDD, W, learningRate = 0.1):
  gradient = augmentedData.map(lambda x: np.dot(x.T, sigmoid(np.dot(x, W)) - y) * class_weight).mean()
  update = np.multiply(gradient,learningRate)
  new_model = W - update
  return new_model
```

Limitations, Challenges, & Future Work

Limited Hyper-parameter Search

- Largely due to time and computing resource constrains
- Exhaustive search should cover larger parameter space for CatBoost

Incompatible Libraries

- LightGBM and CatBoost libraries are not incompatible with each other.
- Wasted a lot of time
- Dropped LightGBM

Future Work

- Refine feature eng/selection methods
- Optimize training time. E.g. partition tuning
- Refine value proposal
- Explore other models

Thank you!