Architecture for an application which predicts likelihood of delay for domestic flights

Adam Letcher, Cyprian Gascoigne, Vicki Foss

INTRODUCTION

This "flight delay forecast application" is designed to predict whether an upcoming domestic flight will experience a departure delay. Given a flight number, our application will look up specific information about that flight (airline, departure and arrival airport, scheduled departure time) as well as current weather information for the departure airport (forecasted daily high and low temperatures, and daily amounts of precipitation and snow). A machine learning model trained on 35 million historical flights and corresponding weather information will then predict the likelihood of a departure delay for that flight. We define a delay as a greater than 10 minute delay at the airport gate.

DEPENDENCIES

This application was built with the following tools:

- An EC2 instance of the Community AMI "UCB W205 Spring 2016", running Linux 2.6 on CentOS 6.6 + Red Hat 4.4.7
- Python 2.7.13, with the following packages installed:
    - requests
    - urllib2
    - json
    - pandas
    - time
    - sklearn
    - numpy
    - unicodedata
- Apache Spark 2.2
- Apache Hive 1.1.0
- Hadoop 2.6
- APIs from http://flightstats.com/ and https://www.wunderground.com/
- Historical weather data from https://www.ncdc.noaa.gov/
- Historical flight data at https://www.transtats.bts.gov/ONTIME/Departures.aspx
    - BTS (Bureau of Transportation Statistics)
- Tableau for data visualizations

The application has not been tested with other versions of the above software, although we can verify that the current scripts will not work with Spark 1.X.

DATA

Historical data sources

- Flights: https://www.transtats.bts.gov/ONTIME/Departures.aspx
- Weather: https://www.ncdc.noaa.gov/

Current data sources

- Flights API: http://flightstats.com/
- Weather API: https://www.wunderground.com/

Notes

- Missing values are coded as -9999.
- PRCP = Precipitation - daily, float (tenths of mm)
- SNOW = Snowfall, daily, float (mm)
- TMAX = Day's Maximum temperature, float (tenths of degrees C)
- TMIN = Day's Minimum temperature, float (tenths of degrees C)

HOW IT WORKS - APPLICATION

Gathering and storing current flight information

Current flight information is pulled from an API hosted by FlightStats, which provides tracking information on all flights departing a given airport at a given time. The FlightStats API responds with data in the json format and includes a large number of features of the flight equipment, the flight route, and positional data on the aircraft. For the initial machine learning model described below, only the data on the flight route (departure airport, arrival airport, departure time, etc.) and the identifying information of the aircraft (i.e. tail number) are used. After querying the API for the top 15 domestic airports by flight volume, the application processes the data using Apache Spark and saves the required dataset, by flight, in an Apache Hive table. This data is then later joined with other related datasets in order to make predictions based on the logistic regression model trained in sci-kit learn.

The two update scripts, flightTracksUpdate.py and flightTracksHiveUpload.py, can easily be modified in the future to expand the scope of airports for which data is collected. The FlightStats API also contains a premium (i.e. paid) feature that would allow for data to simply be collected for all flights as opposed to bucketed by flight or route, which would improve the processing performance of the application.

Gathering and storing current weather information

First, it was necessary to map airport codes to a specific location (latitude and longitude) in order to look up local weather conditions. Using the document "Airport Zip Codes (for weatherscraping).csv" [sourced from Bureau of Transportation Statistics], a Python script was written to retrieve latitudes and longitudes for 395 US airports. Fifteen of the airports were not listed in CSV file, and so location information was retrieved and entered manually. Once correct location information was obtained, a clean data frame was created and saved as "airport_locations.csv", to speed up future weather queries. [See: get_weather_forecasts.ipynb in Github Repo.]

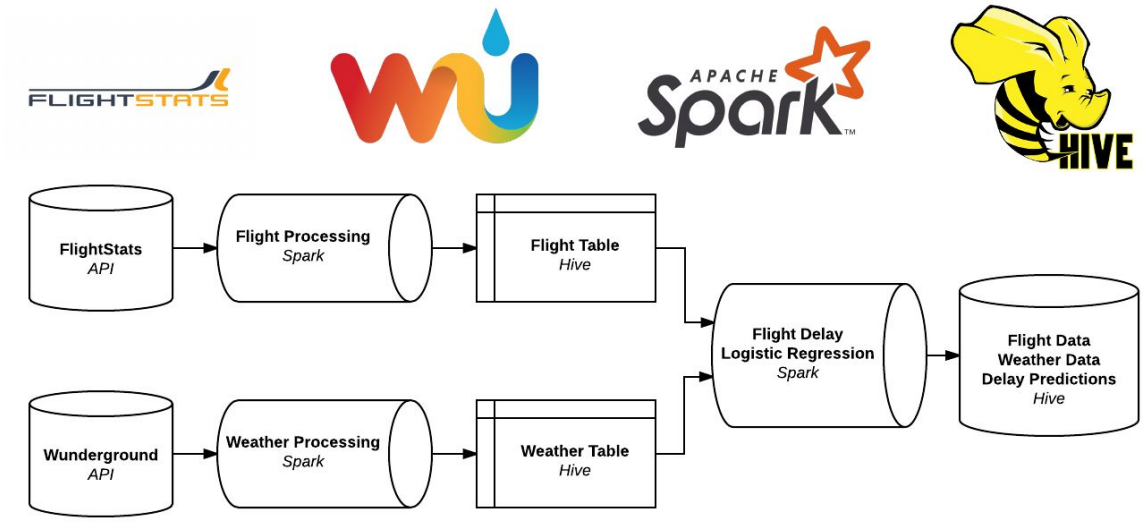
Next, a Python script was created to retrieve weather information for all the airports, on a loop (using the latitude/longitude mappings from the previous step). Because the Weather Underground API limits queries from any one API key to 10 queries per minute, the script pauses for 12 seconds (to be on the safe side) after every query. Since Weather Underground frequently updates weather information, it often happens (especially when rapidly looking up 395 locations) that there are urlopen errors, timeouts or other errors if weather information is being queried at the exact time the source is updating it. Because of this, our script saves those query "failures" in a list and after going through all the airports, looks up the failures once more—and yet a third time for any location where the data still could not be retrieved. The success rate for retrieving weather information for all airports after these three passes is very high (>99%). Weather information from Weather Underground is accessible in JSON format. It is parsed with Python and the weather measures of interest are converted to the same unit measures as the historical weather data

from NOAA. These values are then stored in a Pandas data frame. [See: get_weather_new.py in Github Repo.]

* Note that since we were working on building a machine learning model based on historical information, it was important that the current weather data be recorded the same way. This meant that since we only had daily measurements in the historical data (daily precipitation, daily high and low temperatures, etc.), we retrieved current daily forecasts with those estimations in order to match the historical data as much as possible.

Finally, the data frame is sent to Hive, where it can be combined with flight information and then retrieved to create a delay prediction using the previously created machine learning model. Hive support had to be enabled in Spark (this was done by adding and configuring a file called hive-site.xml to the $SPARK_HOME/conf directory; this was necessary since we upgraded to Spark 2.2 on our instance), and then using the .enableHiveSupport() method on SparkSession in the get_weather_new.py script, allowing it to be able to change the pandas dataframe containing the results of the queried weather information to a Spark data frame which can then be saved as a Hive table when the script is executed via spark-submit.

As demonstrated in the Jupyter Notebook get_weather_forecasts.ipynb, the script could potentially be adapted to collect current weather conditions at all airports (right now it just collects expected daily values), as well as forecasts up to 10 days in the future and the previous day's actual high and low temperatures and total precipitation. Ideally all of this information could be saved in order to (a) use weather forecasts to make a prediction about flight delays further in advance, and (b) to store true historical daily weather values by collecting information about the previous day as well as querying each location various times throughout the day and adding timestamps, so as to feed these precise current weather conditions (instead of daily averages) into the machine learning model, and thus be able to use current weather conditions when making a prediction about a flight delay as well.



SERVING LAYER

The Hive tables `flightstats` and `current_weather_new` are queried, transformed, normalized and finally fed into the ML model. The predictions are stored in a final Hive table called `delaypredictions`.

Visualizing current flight information, weather, and delay predictions via Tableau

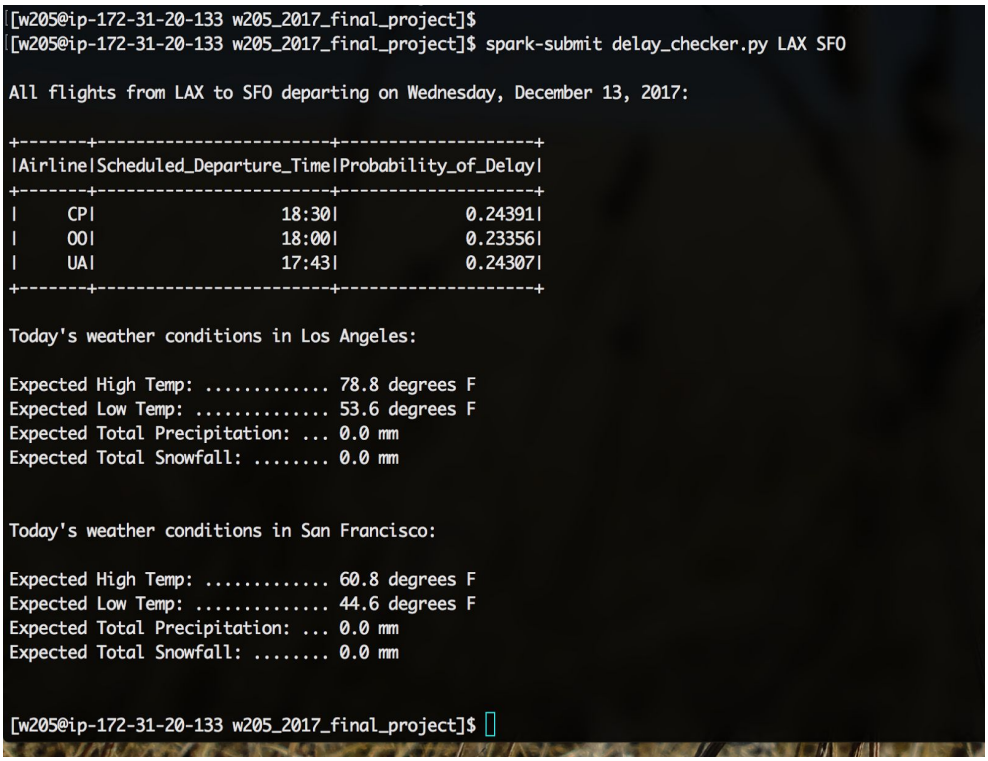
Through Tableau's Cloudera Hadoop support, we are able to connect Tableau directly to the Hive tables in our instance. This allows us to map airport locations, visualize the predictions regarding the probability of delays according to different variables (airport, airline, departure hour, weather variables, etc.) live with current data in our Hive tables. *Note: for this to work the command* `hive --service hiveserver2 &` *must be entered as root user in our EC2 instance.*

Using a pyspark script to display the likelihood of a current flight's delay

For delay predictions on all flights currently being tracked between two particular airports, the Hive table `delaypredictions` is queried using the script delay_checker.py, which can be executed using spark-submit as follows:

```
$ spark-submit delay_checker.py <departure airport> <arrival airport>
```

Local weather information for departure and arrival cities is also displayed. Example:



If no arguments are entered after the command `spark-submit delay_checker.py`, then a table is returned with the 100 flights currently being tracked which are the most likely to be delayed, according to our model.

HOW IT WORKS - HISTORICAL AND MACHINE LEARNING MODEL

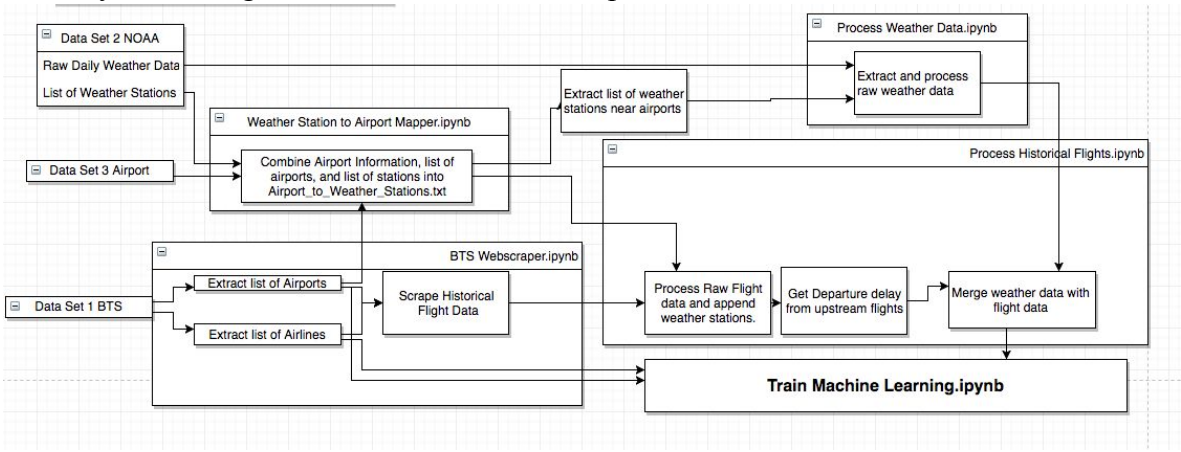
Summary

-https://github.com/kippig/w205_2017_final_project/tree/master/Historical
-Data set 1: All historical domestic flights scraped from BTS website dating back to 1987.
-Data set 2: All historical daily averages from NOAA (dating back to 1862).
-Data set 3: Domestic Airport information (position, address etc) from BTS.
-Training data is a subset, dating from 2005-2017.
-Joined data set 1 to data set 2 by using data set 3 as a map/relation.
-Final pre-processed data is 35 million rows.
-Random sampled data to ascertain max/min values for features to normalize.
-Batch trained on SKLearn using warm-start using a random order of the data.

Data set 1, historical flights data:

First, we scraped the historical flight data with BTS Webscraper.ipynb . This initial scrape set the parameters of what historical data we have available and thus what live data we would try to acquire. This set out our list of domestic airports, airlines and what years we could use on training. By looking at the source elements on the website, we could extract a

list of the airline and the airports. Using Regex these were extracted and then fed into Python's urllib to do an HTTP POST request (in addition to several keys that were required). The HTTP POST request return csv files for all domestic flights post 1987. /csv directory is not on github as the files summed up to size 8.4 GB.



For a close image, please follow: Historical Flight Flowchart

Data set 2, historical weather data and stations:
This set of data is NOAA's daily historical weather information. This was stored and processed here, using Process Weather Data.ipynb . The data was downloaded from NOAA in a tar.gz and extracted. Separately, a list of weather stations was provided in ghcnd-stations.txt.

Data set 3, Domestic Airport information:
Airport Zip Codes (for weatherscraping).csv
This is a .csv acquired with Airport code, runway latitude and longitude. It also has other information we did not use such as address, phone numbers, zip codes etc. We ended up using latitude and longitude instead of zip codes.

Joining the data:
The first step taken was to join weather stations to airports in data set 3. This can be seen in notebook Weather Station to Airport Mapper.ipynb.

In order to join the data first the flight data was preprocessed in Process Historical Flights.ipynb. Each csv of raw historical flight data was only for 1 airline and 1 airport in combination. Additionally, the rows did not have the airport as a value. The first step was to append the departure airport and weather station to each row. After this was done, the weather data was filtered with the weather station data. Finally the weather data and flight data were merged on the weather station column.

Machine Learning:
Train Machine Learning.ipynb

Airlines, airports and tail numbers were label encoded. We normalized the data using range normalization where the max and min for each feature were sampled from a random 1,000,000 rows of the training data. Finally, we trained using SGD Booster and warm start. Training used 500,000 rows per batch for a total of 70 batches.

POTENTIAL FOR SCALE-OUT AND ENHANCEMENT

- Covering a wider range of international flights and regions.
- Adding more features + feature engineering to the machine learning model.
    - Use upstream flight to predict current and future delays.
    - Binarize categorical variables.
    - Train on hourly or minute based weather data in addition to daily.
    - Tune hyperparameters to optimize ML model. (This takes a very long time).
- Optimize ML model for large datasets (sklearn is slow: 10-12 hours per model).
- Updating machine learning model based on new data collected.
- Collecting more granular weather data based on specific times (rather than daily averages) to improve ML model and then use current conditions (rather than forecasts) to make the predictions.
- Improving the collection of weather forecasts in order to improve predictions for flights further into the future (or using historical averages for location/date where a forecast is not available).
- Spark Streaming updates to Hive tables
- Add predictions/data for tarmac delay.
- Implement fuzzy data methods in the historical data.
    - Fill in missing tail numbers, schedules, delays.