Hong Kong University of Science and Technology

MSBD 5003 Project

Group 15

Flight Delay Prediction

Author:
Chen Yuxiang
Wang Jinghe
Xiong Di

December 19, 2017

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Contents

1 Introduction

Flight delay is emerging as a severe problem for travelers. In the United States, the first choice for most short-trip travelers is also the airplane and Generally, evenly a small delay is unaffordable for these short-trip domestic travelers who possibly will have to attend business meetings on time. Under this situation, accurately predicting the time of small flight delay would be quite helpful for travelers requiring punctuality. Intuitively, flight delay shows significantly different patterns with the change of external environment such as weather or time and with the variety of some internal factors such as the type of the airline or departure airport. During the period of time when harsh weather conditions occur, there is a higher possibility of flight delay. Also, some airlines or airports might be poorly managed that these would lead to intrinsic delays that would not vary much with the external environment. Based on these commonsense, it is expected that flight delay could be predicted with history flight data. Through building a regression model on the flight records of the United States in a single year, we aim to offer suggestions for travelers to schedule their trips ahead of time.

2 Data Analysis

2.1 Dataset description

The data we employed comes from kaggle[1]. It mainly contains information of three aspects, all of which is in the format of csv.The first one is named airline.csv, mapping airline identifiers to their names. The second one is named airport.csv, containing information of airports such as location, airport named and airport identifier. The key column of this table is the airport identifier. The third one contains specific flying records and delay time of each flight in the year 2015 of the US and is also the major source we worked with. The columns and the description of each column of this data source is illustrated in Table 1. In this table, the airline column corresponds to the airline identifier in airline.csv and the original and destination airport correspond to the airport identifier in the airport.csv. The identifier of this table is flight number, which marks distinct flights.

2.2 Data Preprocessing

In this part, we describe how we preprocessed the data before analyzing the data and building a prediction model on it. We mainly preprocessed the raw flight record data, which contains lots of redundant attributes and missing values. By leveraging the API provided by spark dataframe, we firstly dropped the attributes that we would not need for further use, and filtered out all record having missing values on any of the attributes that were kept. After this step, we have all the data necessary for building a regression model to predict the time of flight delay. Then we resolved the redundancy in the data. It could easily seen from the schema of the flight record table that the first four attributes are actually referring to the same thing, which is the day of the flights. In the original data, the year, month, day of month of a particular day

Column Name	Description
YEAR	Year of the trip
MONTH	Month of the trip
DAY	Day of the trip
DAY_OF_WEEK	Day of Week of the trip
AIRLINE	Airline Identifier
FLIGHT_NUMBER	Flight Identifier
TAIL_NUMBER	Aircraft Identifier
ORIGINAL_AIRPORT	Starting Airport
DESTINATION_AIRPORT	Destination Airport
SCHEDULED_DEPARTURE_TIME	Planned Departure time
WHEELS_OFF	Time the wheel leaves the ground
TAXI_OUT	Time from departure to the taking off point
DEPARTURE_TIME	WHEELS_OFF-TAXI_OUT
DEPARTURE_DELAY	Total Delay on Departure
DISTANCE	Distance between the two airports
AIR_TIME	Time spent traveling between the two airports
CANCELLED	Mark if the flight is canceled
AIR_SYSTEM_DELAY	Delay caused by air system
SECURITY_DELAY	Delay caused by security
AIRLINE_DELAY	Delay caused by the airline
LATE_AIRPORT_DELAY	Delay caused by aircraft
WEATHER_DELAY	Delay caused by weather

Table 1: Dataset Description

in a year are splitted and considered as three different attributes. Considering the situation that our model was designed for and for the convenience of preprocessing , we combined these three attributes into a single numerical attribute, which denotes how many days have gone by since the first day of the year for a particular day in a year. This scalar actually contains the encoded information of month and day of month. These two could be derived from the newly generated numerical attribute and the machine learning models are expected to decode this attribute easily.We did not consider the year since our model was trained on the flight data of 2015 and it did not aim to reflect the relationship between year and the time of delay. With respect to the fourth column, which denotes which day of a week each day belongs to, we kept it. Although this could be calculated based on the numerical attribute we just generated, it might be strongly correlated with the time of delay since intuitively, people might be more likely to travel on weekends and there might be more delays on weekends. We wanted to keep to the feature selection stage to test whether it is a predictive attribute. For the attribute named SCEDULED_DEPARTURE_TIME, we used similar technique as above to transform it from time to a scaler value, which means how many minutes have been since the first minute of the day. The attributes that we kept after the data preprocessing are shown in Table 2. In the table, the first four are candidates for features and the last one, the departure delay, serves as the label.

Column Name	Description
NEW_DAY	Day of year since the first day
ORIGINAL_AIRPORT	Month of the trip
DESTINATION_AIRPORT	Destination Airport
DAY_OF_WEEK	Day of Week of the trip
NEW_SCHEDULED_TIME	Minute of day
DEPARTURE_DELAY	Total Delay on Departure

Table 2: Selected Attributes

2.3 Feature Selection

In order to obtain better predicting results, we need to choose some proper features from tens of attributes included in the dataset.

As is shown in Fig.1(a), there is a strong relationship between delay and scheduled departure time in a day.The delay around 5 to 6am is lower and it reaches a peak at around 8 to 9pm.(the peak at 2 to 3am is regarded as outliers)

Then, we focus on relationship between delay and day of week(Fig.1(b)). The link is not obvious, so we ignore this feature in later analysis.

But there is a obvious association between delay and day in year. As we can see from Fig.1(c), the delay fluctuate a a lot during the whole year. However, we cannot see the type of relation intuitively from the figure.

Similarly, different airlines have different delay time(Fig.1(d)) and we cannot observe the type of regression. Because of relatively less airline, we can build models for each airline individually.

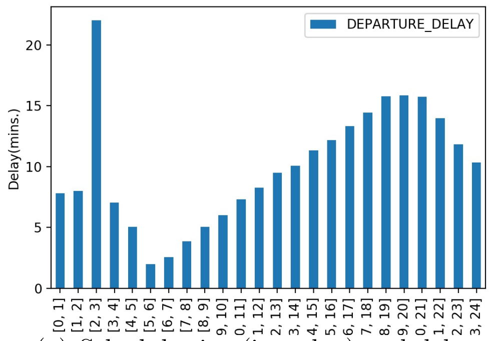
Next, We want to analyze the relationship between origin airport and delay. As is shown in Fig.1(e), airport truly has effect on delay but it is not distinct. Besides, it impossible to build model individually for more than 300 airports.So, we try to group them and do some correlation analysis(Fig.1(f)). The Pearson correlation coefficient is very low between grouped airports and delay. Consequently, we treat this feature a categorical attribute in our model. Finally, we choose six features(Table 2) to build our regression model.

3 Model Construction

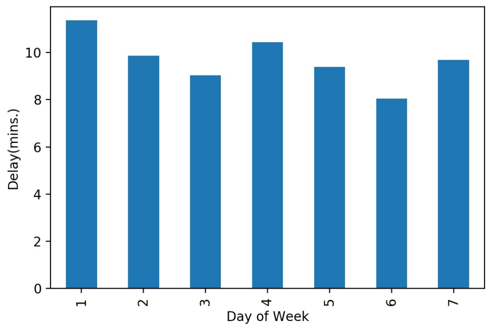
3.1 Model Overview

We use the features dominantly important described in last chapter to train our machine learning models. the whole procedure of model construction is divided into two phase: training phase and prediction phase.

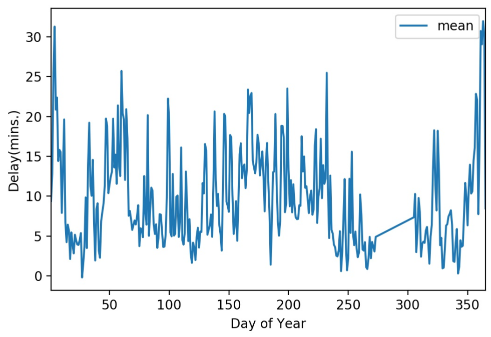
In the training phase, We would apply several transformers and estimators included in pyspark.ml library to features to prepare the input data. And then processed feature vectors and labels would be send into our machine learning model constructed by MLlib library. Both pipeline models and machine learning models would be saved as off-line models for latter use. The phase requires lots of time to finish and we have to train the



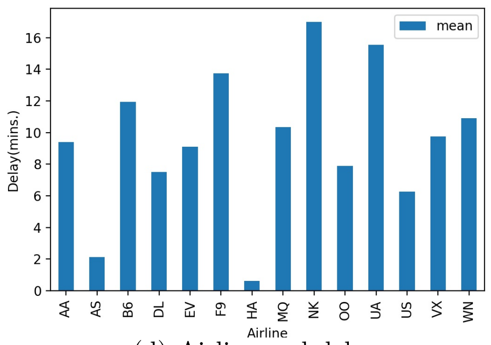
(a) Schedule time(in a day) and delay



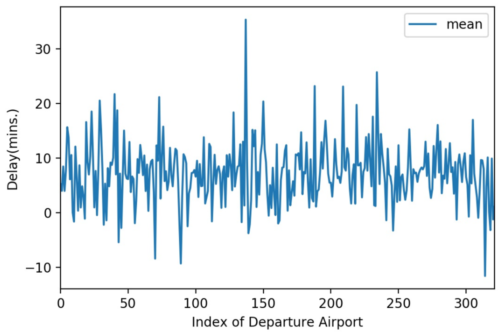
(b) Day of week and delay



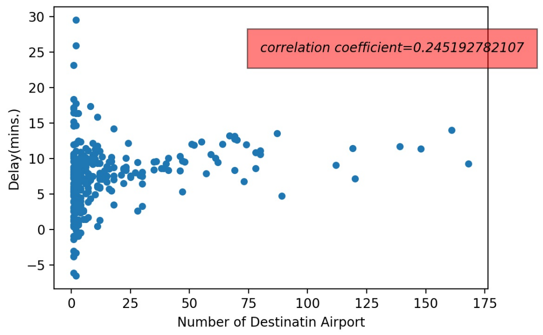
(c) Day in year and delay



(d) Airline and delay



(e) Origin airport and delay



(f) Grouped(by number of destination airport) origin airport and delay

Figure 1: Relationship Analysis Between Delay and Some Features

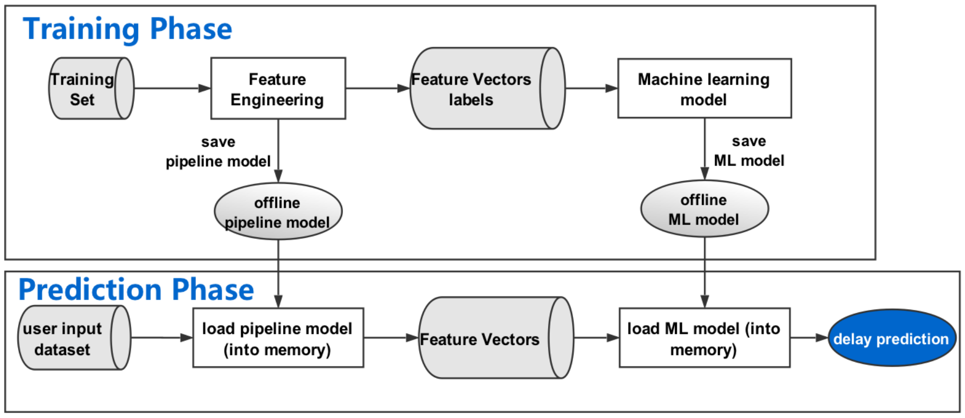


Figure 2: Model overview.

model in advance and use the prepared models directly in the next model, decreasing the customers' waiting time and increasing the production competition.

For the next prediction phase, we would used trained models to transform on-line inquiring data to feature vectors and send it into models already trained and loaded into memory in advance to do quick prediction. For each airline, we build a separated model to perform regression task to predict the delay time.

3.2 Pipeline Model

It is very essential to prepare the data for prediction model. All features would be separated into three types: categorical features, numeric features, labels.

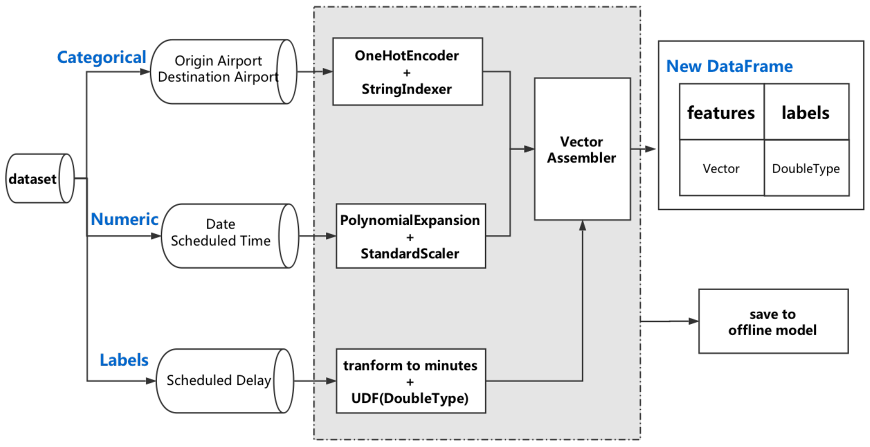


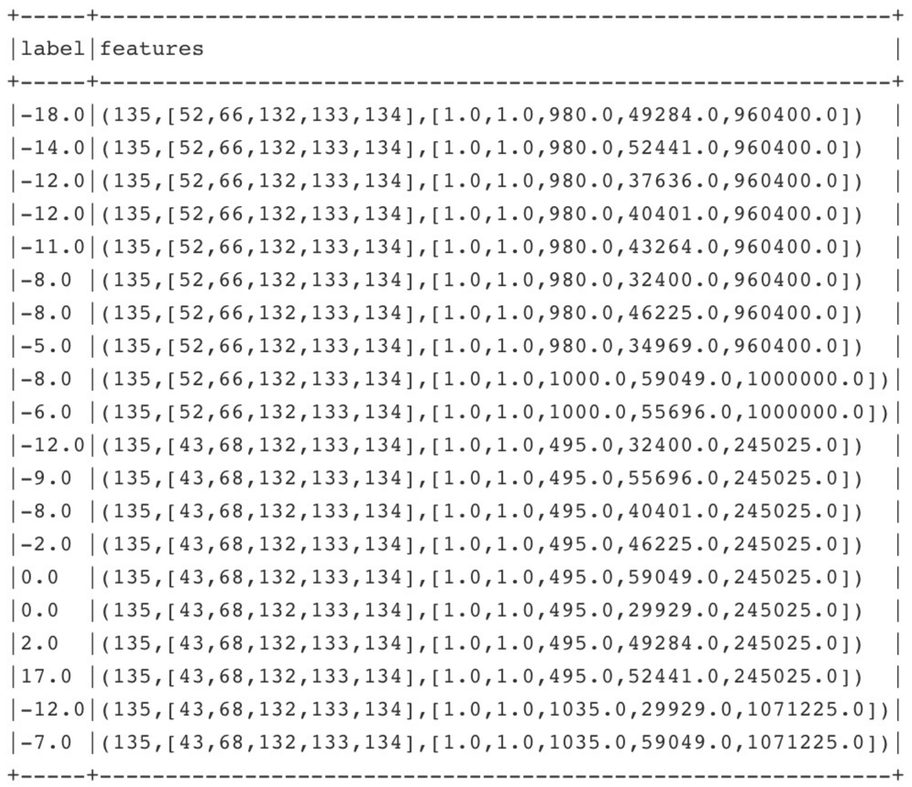
Figure 3: Pipeline model.

We have two features belonging to categorical data, ”Original Airport” and ”Destination Airport”. Two features would be sent into transformers like ”OneHotEncoder” to implement one-hot transformation and ”StringIndexer” to give correct index name to each processed features. These two transformers ensure that all categorical features would be ready for machine learning model.

For Numeric features, We have ”Date” and ”Scheduled time” two features. according to the Exploratory analysis above, we have notice that the square value of Scheduled time would have strong relationship with our label. So we use a powerful transformer called ”PolynomialExpansion” to calculate the squared value, which would be used as our new feature. It is a indispensable step to implement standardization before using all kinds of numeric data. These numeric features would be standardized by a estimator called ”StandardScaler”.

”Scheduled delay” is our label and it is very necessary to transform it into values that are measured by minutes. We design UDF function to change its type from integer into double type, under which condition that we can compute MSE and RMSE conveniently and directly.

All features and labels would be assembled together by a transformer called ”VectorAssembler”, the output of which is a new Spark DataFrame containing only two columns: features and labels. It is very obvious to tell the structures and meanings it demonstrate.



```
+-----+-------------------------------------------------------------+
|label|features                                                     |
+-----+-------------------------------------------------------------+
|-18.0|(135,[52,66,132,133,134],[1.0,1.0,980.0,49284.0,960400.0])   |
|-14.0|(135,[52,66,132,133,134],[1.0,1.0,980.0,52441.0,960400.0])   |
|-12.0|(135,[52,66,132,133,134],[1.0,1.0,980.0,37636.0,960400.0])   |
|-12.0|(135,[52,66,132,133,134],[1.0,1.0,980.0,40401.0,960400.0])   |
|-11.0|(135,[52,66,132,133,134],[1.0,1.0,980.0,43264.0,960400.0])   |
|-8.0 |(135,[52,66,132,133,134],[1.0,1.0,980.0,32400.0,960400.0])   |
|-8.0 |(135,[52,66,132,133,134],[1.0,1.0,980.0,46225.0,960400.0])   |
|-5.0 |(135,[52,66,132,133,134],[1.0,1.0,980.0,34969.0,960400.0])   |
|-8.0 |(135,[52,66,132,133,134],[1.0,1.0,1000.0,59049.0,1000000.0])|
|-6.0 |(135,[52,66,132,133,134],[1.0,1.0,1000.0,55696.0,1000000.0])|
|-12.0|(135,[43,68,132,133,134],[1.0,1.0,495.0,32400.0,245025.0])   |
|-9.0 |(135,[43,68,132,133,134],[1.0,1.0,495.0,55696.0,245025.0])   |
|-8.0 |(135,[43,68,132,133,134],[1.0,1.0,495.0,40401.0,245025.0])   |
|-2.0 |(135,[43,68,132,133,134],[1.0,1.0,495.0,46225.0,245025.0])   |
|0.0  |(135,[43,68,132,133,134],[1.0,1.0,495.0,59049.0,245025.0])   |
|0.0  |(135,[43,68,132,133,134],[1.0,1.0,495.0,29929.0,245025.0])   |
|2.0  |(135,[43,68,132,133,134],[1.0,1.0,495.0,49284.0,245025.0])   |
|17.0 |(135,[43,68,132,133,134],[1.0,1.0,495.0,52441.0,245025.0])   |
|-12.0|(135,[43,68,132,133,134],[1.0,1.0,1035.0,29929.0,1071225.0])|
|-7.0 |(135,[43,68,132,133,134],[1.0,1.0,1035.0,59049.0,1071225.0])|
+-----+-------------------------------------------------------------+
```

Figure 4: DataFrame containing transformed data.

We should attach more attention to the next step: assemble all transformers and estimators, stored as stages, into a completed pipeline model. the pyspark library provides the mutual function to store the model in files. When we do the prediction, it is necessary to apply this pipeline model loaded into memory in advance to input inquiring data and extract prepared features and labels. Some parameters such like the standard deviation in StandardScaler are trained by training dataset, which make it impossible to train the pipeline model on-line, consuming pretty of user time. So we select the pretrained pipeline models to speed up the process of on-line feature transformation.

3.3 Model Tuning and Selection

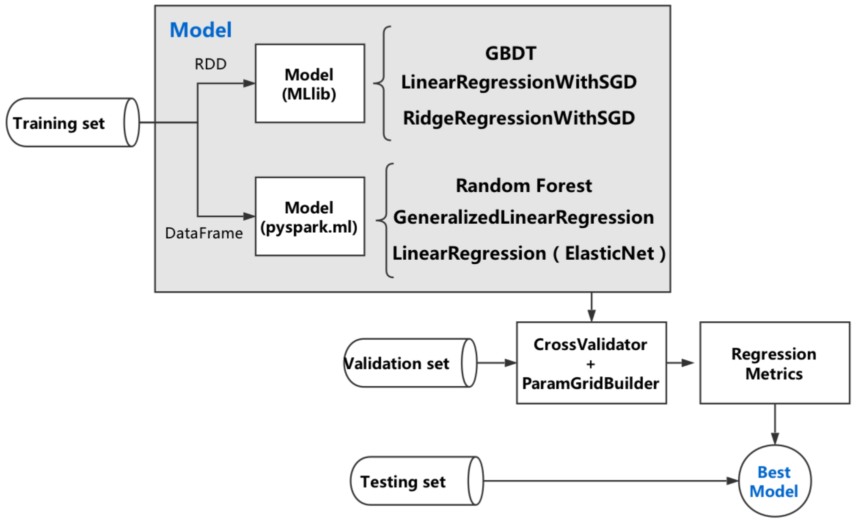
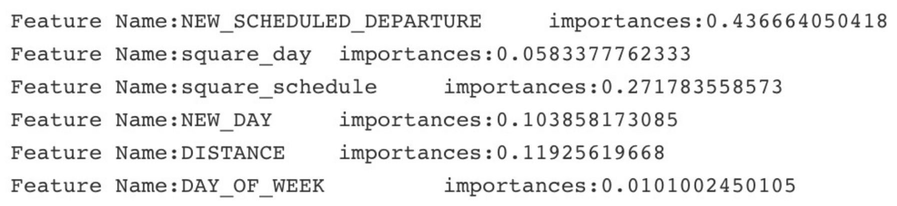


Figure 5: Model tuning.

We have tried building models by pyspark.ml and MLlib library separately. And we figure out the difference between two types of models. We use training dataset to train the various models and compare their performance on validation dataset to fine-tune their hyper-parameters. finally we would choose best model based on their performance on testing dataset.

We used pyspark.ml library firstly. The high-level integrated DataFrame API makes it convenient to send assembled vectors into models. Some excellent regression model such as "GeneralizedLinearRegression" and "LinearRegression with Elastic-net" contain outstanding generalization ability. And the Random Forest model provided by this library would output the features importances. It assists us to drop unimportant features to avoid introducing noise into prediction models.

In next step, we use MLlib to construct models. It should be noticed that data stored in DataFrame should transform into RDD before send into models. In fact, the



```
Feature Name:NEW_SCHEDULED_DEPARTURE     importances:0.436664050418
Feature Name:square_day  importances:0.0583377762333
Feature Name:square_schedule     importances:0.271783558573
Feature Name:NEW_DAY     importances:0.103858173085
Feature Name:DISTANCE    importances:0.11925619668
Feature Name:DAY_OF_WEEK         importances:0.0101002450105
```

Figure 6: feature importances generated by Random Forest model.

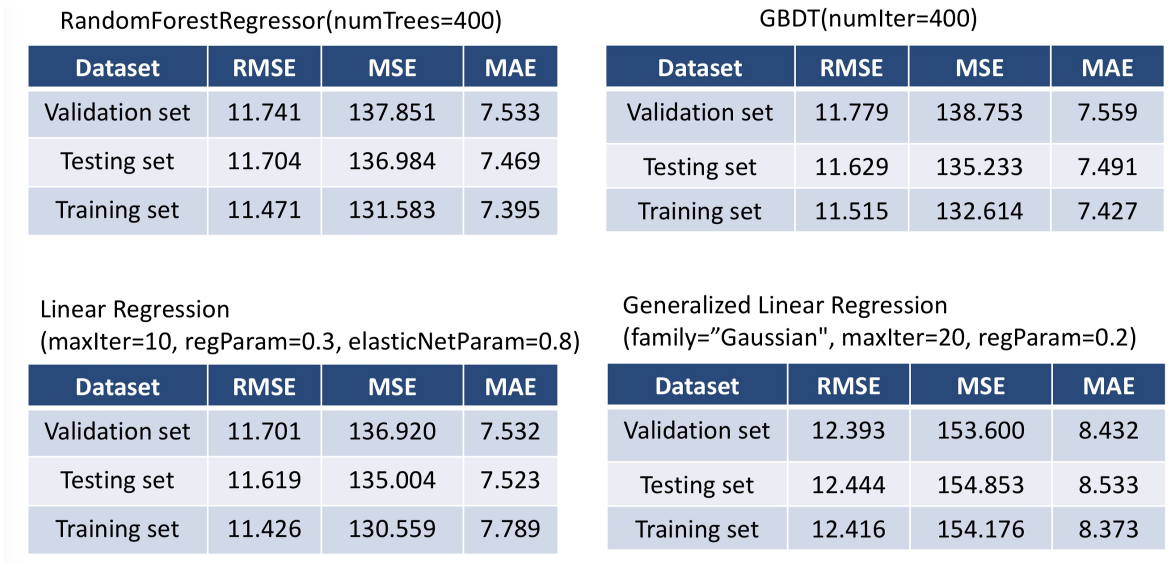
process of transform vectors stored into RDD contains some bugs written in documentation and it is not easy to implement it. It is important to tell the difference between the sparse vectors and dense vectors. And after that the data should be labeled with another type called "Labelpoint". Fortunately for models built by MLlib, they can be optimized by stochastic gradient descent algorithm, Which obviously speed up the procedure of training models. Some magic models such as "Ridge linear regression" and ensemble methods "GBDT" are provided by MLlib. Between so many machine learning models provided by MLlib, GBDT models require longest time to train, due to the idea of Boosting cannot implemented in parallelized way. Therefore, it would take more computation sources and stages in spark to decrease the cost defined by MSE.

When trying two library, we figure out some differences between them. pyspark.ml is maturer and it is easy to call its inner function or classes to perform machine learning tasks. It also has a powerful class called "CrossValidator" can implement cross validation easily to tune hyper-parameters. So It is convincing that we should give it priority. But for MLlib, though it has some unfixed bugs described in Documentation, it totally speeds up the process of training models, and thousands of senior engineers working in Spark would focus on the improvement of MLlib, And it has a promising future and already demonstrate its magic training speed. MLlib is a recently heated topic in related technical community.

3.4 Model Performances

We choose MSE(mean square error), RMSE(root square error), MAE(mean absolute error) to measure the fitting ability of our regression models.

We list all metrics for all excellent models we trained on validation dataset, testing dataset, training dataset separately. We find that though we took lots of computation sources and time to train the GBDT model, the Linear Regression model beat it in testing dataset. Therefore, it is reasonable to believe that the elastic net can provide magical generalization ability to this model with less complexity. Only take iteration 10, it can get less MSE, which means that it contains excellent fitting ability. It also demonstrates that from the prospective of statistical knowledge, the regression tasks seems to agree with the assumption of less flexibility model.



RandomForestRegressor(numTrees=400)

Dataset	RMSE	MSE	MAE
Validation set	11.741	137.851	7.533
Testing set	11.704	136.984	7.469
Training set	11.471	131.583	7.395

GBDT(numIter=400)

Dataset	RMSE	MSE	MAE
Validation set	11.779	138.753	7.559
Testing set	11.629	135.233	7.491
Training set	11.515	132.614	7.427

Linear Regression
(maxIter=10, regParam=0.3, elasticNetParam=0.8)

Dataset	RMSE	MSE	MAE
Validation set	11.701	136.920	7.532
Testing set	11.619	135.004	7.523
Training set	11.426	130.559	7.789

Generalized Linear Regression
(family="Gaussian", maxIter=20, regParam=0.2)

Dataset	RMSE	MSE	MAE
Validation set	12.393	153.600	8.432
Testing set	12.444	154.853	8.533
Training set	12.416	154.176	8.373

Figure 7: Comparison of model performances.

3.5 Prediction Phase

in the prediction phase, we would load the pipeline model and machine learning models pre-trained into memory in our web server. When there is a customers' data coming, we would send 14 copies of personal data into 14 pipeline models built for 14 different airlines. And 14 feature vectors would be sent into corresponding machine learning models to perform regression tasks. The 14 numbers representing predicted delayed time would be return into server, which would be used as references for customers to decide which airlines to take, avoiding waiting for long time in airports[2].

4 Web Service

4.1 brief introduction of Azure web service

Azure App Service is provided by Azure to enable developers to build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Developers are free to build apps or websites with any of their preferred programming languages and frameworks, deploy them to Azure web server using an uniformed interface without the need to take into consideration such problems as different project structures or different requirement caused by the variety of developing environments. Besides, Azure App Service is fully-managed in the sense that it provides powerful infrastructure maintenance and load balancing. It supports users to add customer domain and integrate a rich variety of identity services into their applications. Apart from the convenience in deployment and the strength in maintenance, Azure also provides both a public and a private IP address for virtual machines, thus making it more efficient for the machines in the back end to communicate with each other and collaborate on the computation tasks received from the user end. These privileges of Azure web service make it the

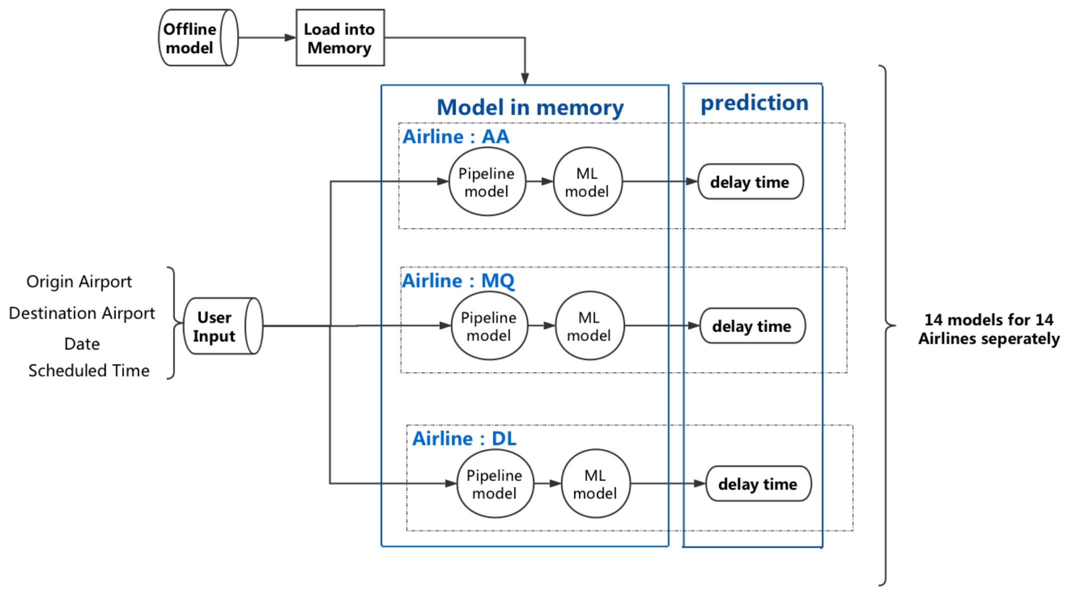


Figure 8: Online prediction overview.

optimal choice for deploying our websites to the cloud.

4.2 Structure of our website

This part illustrates the structure of the whole web service and concentrates on the data flow among the user end, the web server and a cluster of virtual machines which carries out the main jobs. Figure 9 shows the major structure of the whole website. As shown in the graph, the whole website consists of three parts in terms of function and is organized based on the MVC structure. The front end contains the source web files and scripts which capture the actions triggered by users, make them structured by transforming them to the format of json and prompt them to the web server for further processing. It serves as the the 'V', namely the view layer in the website. The second part, Azure web server, serves as the 'C' layer, namely the controller. It could be viewed as a bridge linking the front end and the model layer. The web server stores source scripts which could receive the json data from the front end and transform the data so that they could be directly taken as input by the model layer. Finally, the model layer, which is composed of a cluster of virtual machines, is responsible for doing the computation and returning the result, which is also organized in the format of json, to the controller layer.

4.3 Implementation of our website

Based on the description of the structure of the website in the previous part, this part will illustrate how we implemented the above website with the Flask framework

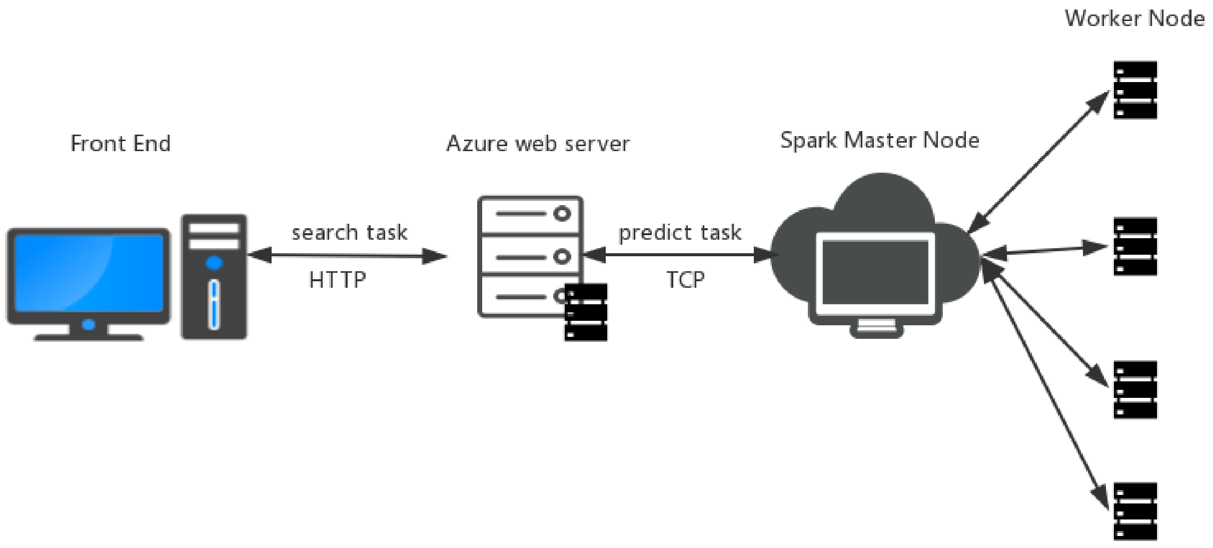
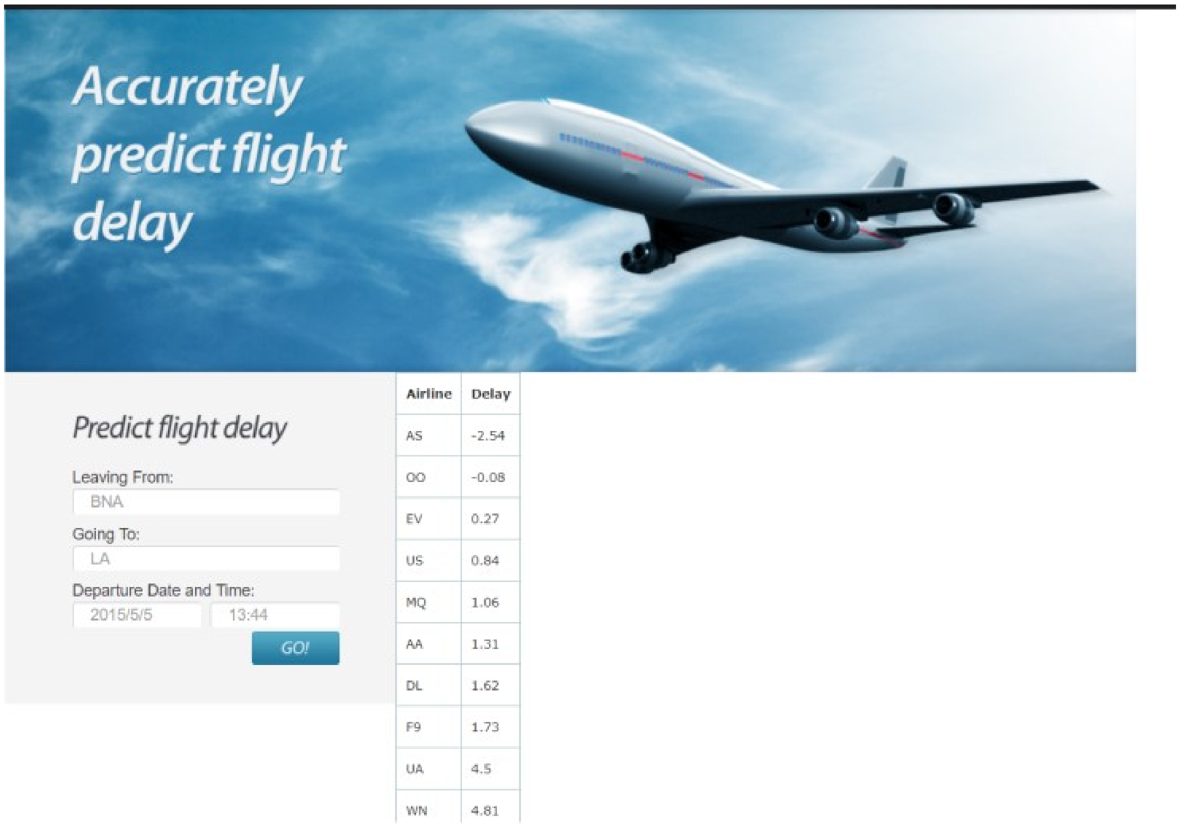


Figure 9: Website Structure

of python. The communication between the front end and the web server is through the standard HTTP protocol and we took advantage of the API provided by Flask[3] to simplify this process. As illustrated above, the web server is considered as the controller layer and its main task is to redirect the computational tasks to the cluster and gather results returned from the cluster, this redirector was implemented in a single python file. For the communication between the web server and the cluster which did the real computation, we leveraged a lower level protocol, the original TCP protocol, which was implemented with the socket API provided in python. In this end-to-end communication, the master node of the cluster serves as the socket server and the web server serves as the socket client. The master node would parse the content containing the tasks to be carried out from the socket stream and distribute them to the worker nodes. The computational task was implemented in pyspark.

In our website, the computational task is to predict the time of delay by the expected departure time of a traveler and his route of travel. These parameters are exchanged between functional parts in the format of json and for the prediction task, we stored the 14 pre-trained pipeline models corresponding to all the 14 airlines in HDFS and the worker nodes feed the parameters to the models and obtain the prediction result, which means the potential time of delay. Figure 10 is a snapshot of the main page of our website. The left part is a web form for travelers to set the key parameters and the table on the right displays the result showing the predicted delay time of each airline. In this table, each row represents an airline and its predicted time of delay under the particular setting of departure time, original airport and destination airport and the airlines are listed in the ascending order of predicted time delay. The time it takes from clicking on the search button and having the result displayed varies between 2 to 5 seconds depending on the expected departure airport a user sets. Generally speaking, a busier airport means more airlines to choose from and thus slightly longer time of computation because we need to feed the data to the models of all airlines that have flights of the appointed route. Furthermore, to save time for unnecessary IO, we loaded all the models to memory on starting the machine. In conclusion, the main function of our website is to offer advice for travelers to choose an airline once their approximate departure time and the exact traveling route is fixed.



Leaving From: BNA

Going To: LA

Departure Date and Time: 2015/5/5 13:44

Airline	Delay
AS	-2.54
OO	-0.08
EV	0.27
US	0.84
MQ	1.06
AA	1.31
DL	1.62
F9	1.73
UA	4.5
WN	4.81

Figure 10: Website Screenshot

4.4 Deployment on Azure Cloud

After running our website successfully on local machine, we need to deploy it on azure cloud. In general, we need two servers to provide a web service on cloud. One is a web server which contains the website. The other is calculation server which undertakes the calculation task. In our project, the first one is provided by Azure App Service and the last one is a virtual machine on Azure.The steps to deploy our flight delay service on Azure will be illustrated as follows.

4.4.1 Web Server

We can begin with deploying our websites on Azure App Service.

Step 1: Create a resource group for all resources we need.

Step 2: Create a App Service with Flask container. Because the website has very low amount of visiting(Currently only us will visit this website), we choose the relatively low service plan(B1 Basic). It can be easily scaled up when there are more hits.

Step 3: Use Git to push our local repository to web server and assure it can be accessed by the public IP or URL.

Step 4: Create Virtual Networks to enable faster communication between web service and spark server.

4.4.2 Spark Server

The next step is to create a Spark server to receive the messages from web server, make prediction and return the results to web server.

Step 1: Create a Ubuntu Virtual Machine for Data Science in my resource group, which pre-installs basic packages and tools including Spark. We choose Basic A3 plan because it needs relatively more power to make prediction.

Step 2: Upload trained model files and Python script for prediction which includes an API for calling the predict function.

Step 3: Upload server script for communicating with web server and calling the predict function.

Step 4: Open the inbound port 9999 for receive messages. Set the local IP address for Spark server and web server respectively.

Step 5: Run the server script to monitor the message from web server. At the moment, it will setup Spark context and load model to RAM(Random Access Memory) in order to save the time of setup and load for every query request.

4.4.3 Performance Analysis

Test Environment:
Spark server: 4 cores CPU, 7GB memory
Web server: 1 core CPU, 1.75GB memory

As we can see in Table 3, the setup time(including Spark setup and model loading) is extremely time-consuming. The waiting time for one query is relatively acceptable,

Table 3: Performance Statistics

Items	Time
Server Setup	about 2 minutes
Average Query Response	about 8 seconds
Average Memory Consumption	2.4GB
Highest CPU Usage	95%
Idle CPU Usage	33%

only at the cost of 35% CPU usage. The detail performance of VM is shown as Fig.11. It is handling requests during 9:15pm to 9:35pm and stay idle later.

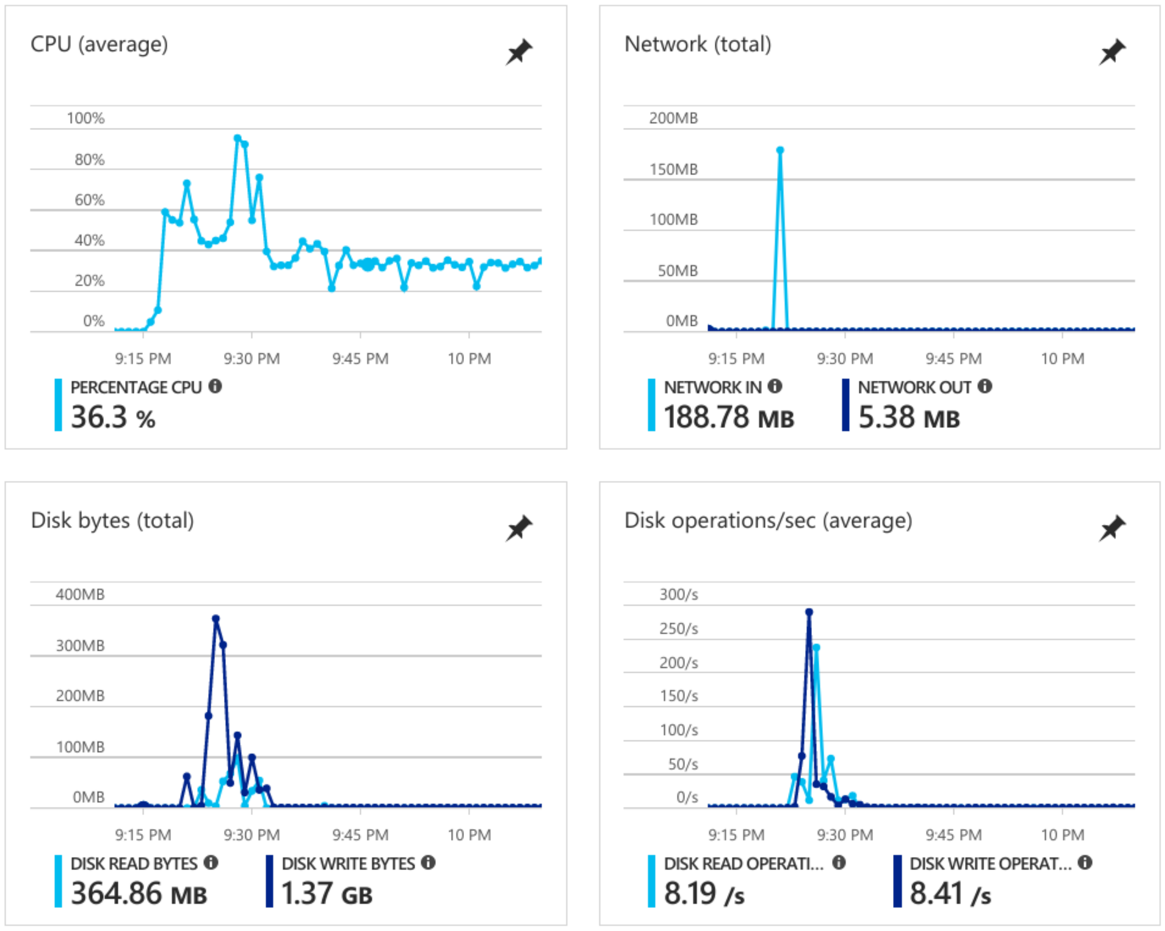


Figure 11: Running Status of Spark Server

5 Conclusion

5.1 RDD

RDD(Resilient Distributed Datasets) is the central data structure of Apache Spark. In order to train the models constructed by MLlib, it is necessary to transform the data from DataFrame into RDD in advance. by using RDD inner supporting functions, we can implement filter, map, flatMap, join operation, Which would assist speeding up the process of calculation of MSE. It can be persisted and released as needed. comparing with MapReduce system built by Hadoop, it used the data cached in memory to decrease computing time.

5.2 Spark ML

In our project, we have tried using both pyspark.ml(the spark library based on DataFrame-API) and MLlib(the new library supported by Spark and based on RDD-API). By these machine learning library, it is more convenient to construct machine learning models in parallelized stages, divided automatically by spark jobTracker in master node. It totally speeds up the training phase of building machine learning models to tackle scalability problems. With optimizers such like Stochastic gradient descent provided by MLlib, We can decrease the cost value quickly.

5.3 Spark SQL

When We do feature engineering, We use Spark SQL to implement feature transformation and other operation. Its Interoperability and standardization convinces majority of users in Apache community, And in documentation, it announces that Many Spark libraries would move to Spark SQL API in the future. the high reliability guarantees the perfect user experience and the similarity with pandas DataFrame decreases the process of getting-start for those Python coders. People can transform the data from Spark SQL to pandas DataFrame easily to do the exploratory data analysis, with high scalability provided by Spark inner optimized functions.

5.4 Azure App Service

Azure app service provides powerful infrastructure maintenance and load balancing for individual users and enterprises to run and scale their applications easily on multiple platforms. It has such a privilege of compatibility that we could build and test our websites in Flask framework locally and then effortlessly deploy it to Azure through just a few lines of git commands. Furthermore, the high quality performance of Azure web server makes it no difference between running locally and on the cloud. There is virtually no delay caused by the response of the server. Besides, Azure web server could communicate with virtual machines carrying out spark computation tasks via internal network, thus further reducing the reaction time from the users' perspective.

In all, we benefit from both the app service and the computing resource provided by Azure.

5.5 Future Work

We have done the prediction of departure delay. In later work, we will finish the prediction of arrival delay. Besides, more features can be considered to predict delay time more precisely, such as distance, arrival time, etc.

In terms of web service, we will provide more information about the query. For example, historical delay time can be shown together with predicted delay. we can build a SQL database on Azure to save the historical and new coming-in data we needed, which can be used to provide historical data for front end display as well as update prediction model.

Besides, we should pay attention to the response time of every request.Some possible optimizations are using more cores in Spark server, using SSD instead of HDD and improving algorithm efficiency.

Github

Our source code can be accessed on Github.
`https://github.com/xiongdi94/no_rush`
`https://github.com/xiongdi94/norush_web`

References

[1] Kaggle flight delay prediction. `https://www.kaggle.com/usdot/flight-delays/data`.

[2] Related tutorial in jupyter notebook format. `https://github.com/Sally68/Flight-delay-prediction-with-spark/blob/master/Flight%20delay%20prediction%20%20I.ipynb`.

[3] A microframework based on werkzeug. `https://github.com/pallets/flask`.