Predicting flight delays using SparkR and H2O

- Tina Wang
- Vaishnavi Jala
- Yuefeng Pan
- Surag Gupta

Introduction

The airline industry in US is totally a red ocean, there are so many airline carriers competing for very less margin, and new entrants keep increasing. Just like the quotes from CEO of American Airlines, Robert L. Crandall, “This is a nasty, rotten business”.

In order to stay in the game, airline carriers are constantly striving to improve in many aspects such as customer services, price, food and so on. Among all the complaints from customers, airline delay is obviously the biggest issue, which can potentially be very costly to customers and airline carriers. In fact, with the increasing number of business and leisure travellers, flight delays are becoming a critical issue every airline carrier faces. Carriers who can accurately predict flight delay and plan flights and logistics accordingly will find huge competitive advantages over others.

Our group is focusing on predicting the likelihood of flight delays through classification techniques. We are also proposing a methodology to predict the delay at each airport and for each carrier. Through accurate classification and numerical predictions, we can help carriers to manage delays ahead of time and save a lot of unexpected losses. This will enable carriers to plan flights in a smart way and optimize logistics and pricing for flights.

Data processing and analysis using Big Data tools

Data collection

We procured the data from the US Bureau of Transportation Statistics (http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time) website. The dataset contains information of all flights in the US, ranging from 1 January, 2014 to 31st December 2015 (2 years).

The dataset consists of 11,627,656 rows, with 28 attributes

Attribute	Field / Description
Year	Year of flight
Month	Month of flight
DayofMonth	Day of month
DayOfWeek	Day of the week (in number)
FlightDate	Date on which the flight took place
UniqueCarrier	Unique Carrier Code. When the same code has been used by multiple carriers, a numeric suffix is used for earlier users, for example, PA, PA(1), PA(2).
Carrier	Code assigned by IATA and commonly used to identify a carrier. As the same code may have been assigned to different carriers over time, the code is not always unique.
OriginAirportID	Origin Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport.
Origin	Origin Airport
OriginState	Origin Airport, State Code
DestAirportID	Destination Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport
Destination	Destination Airport
DestState	Destination Airport, State Code
DepTime	Actual Departure Time (local time: hhmm)
DepDelay	Difference in minutes between scheduled and actual departure time. Early departures show negative numbers.
ArrTime	Actual Arrival Time (local time: hhmm)
Cancelled	Cancelled Flight Indicator (1=Yes)
Distance	Distance between airports (miles)
CancellationCode	Specifies The Reason For Cancellation
CarrierDelay	Carrier Delay, in Minutes Weather Delay, in Minutes
WeatherDelay	Weather DelayLate Aircraft Delay, in Minutes, in Minutes
NASDelay	National Air System Delay, in Minutes
SecurityDelay	Security Delay, in Minutes
LateAircraftDelay	Late Aircraft Delay, in Minutes
FirstDepTime	First Gate Departure Time at Origin Airport
TotalAddGTime	Total Ground Time Away from Gate for Gate Return or Cancelled Flight
LongestAddGTime	Longest Time Away from Gate for Gate Return or Cancelled Flight

The files were available in the form of individual CSV files for each month, thus adding up to 24 CSV files for 2 years of data.

Data processing

The data we procured had inconsistencies, and therefore, we had to process the data prior to performing any kind of analysis on it. Dealing with close to 11.6 million rows, it was necessary that we use a tool that handle such a huge volume of data.

The data was too large for our regular 8GB computer to handle. Therefore, we uploaded the individual CSV files to an Amazon S# bucket on AWS. We decided to use Apache Hive to process our data, and hence, we created a Hadoop cluster on AWS Elastic MapReduce, with the following configuration:

```
Vendor: Amazon
Release: emr-5.0.3

Software:
Hadoop 2.7.3
Hive 2.1.0

Hardware: 1 vCPU, 3.75GB RAM, 410GB HDD
```

We used Apache Hive to process the data, and performed cleaning steps such as eliminating quotation marks and creating derived columns, based on our necessities for anaysis.

Setting up EC2 instance on AWS

Once we had the data hosted on the S3 bucket, we created an Amazon EC2 (Elastic Compute Cloud) instance, with a `t2.2xlarge` configuration, which would help us in processing and analyzing the large dataset we had. Additionally, we installed RStudio Server and RShiny on the instance to enable analysis on our dataset. The follwing configuration was used to facilitate this:

```
#!/bin/bash
#install R
yum install -y R
#install RStudio-Server

wget https://download2.rstudio.org/rstudio-server-rhel-0.99.465-x86_64.rpm
yum install -y --nogpgcheck rstudio-server-rhel-0.99.465-x86_64.rpm

#install shiny and shiny-server
R -e "install.packages('shiny', repos='http://cran.rstudio.com/')"
wget https://s3.amazonaws.com/rstudio-server/rstudio-server-rhel5-1.0.44-x86_64.rpm
yum install --nogpgcheck rstudio-server-rhel5-1.0.44-x86_64.rpm

#add user(s)
useradd surag0107
echo surag0107:password | chpasswd
```

Since the OS for the instance was CentOS 5, installation of certain packages in R was required to be run on the unix CLI. Firstly, we required the `OpenSSL` package to be installed. For this, we SSH into the EC2 instance using Putty. We then install `OpenSSL` and `devtools`, with the below command:

```
$ sudo yum install curl-devel
```

We then log onto the RStudio Server using the Public DNS

Initializing Spark and H2O on RStudio Server

For our analysis, we decided to use Spark, in conjunction with H2O and mlLib. We do this by initialzing a spark connection in R, as shown below:

```
library(sparklyr)
library(h2o)

spark_install(version = "2.0.0")
conf <- spark_config()
conf[["spark.sql.warehouse.dir"]] <- "/dev/shm"

options(rsparkling.sparklingwater.version = '1.6.8')
sc <- spark_connect(master = "local",spark_home = Sys.getenv("SPARK_HOME"), config
= conf)

##Initialize H2O
h2o.init(nthreads = -1)
```

Goal

The goal of our analysis was two-fold:

i. Predict whether a flight's departure would get delayed by 15 minutes or more for a given instance (Classification) ii. Predict the delay in departure of a flight in minutes (Regression)

For this, the attributes present in the original dataset were used. The dataset was split into training and test instances for the purpose of model training and evaluation.

Basic data aggregation and processing were done using `sparklyr`. For more information on how to use `sparklyr`, refer to this link (http://spark.rstudio.com/)

Introduction to H2O

H2O by Oxdata brings better algorithms to big data. H2O is an open source math & machine learning platform for speed and scale. H2O enables enterprises to use all of their data (instead of sampling) in real-time for better predictions.

Companies such as Capital One, TransAmerica, Hospital Corporation of America, Progressive Inc. use H2O for predictive analytics and building recommendation platforms using Big data

For our use case, we chose H2O as our primary platform since it enables us to use and compare various predictive algorithms, to ensure high accuracy in our final model. It also provides us with the flexibility of using R, a tool that most data scientists are familiar with. At the same time, it has more APIs which we can connect to such as R, Python and JSON. In addition to those, we've noticed that H2O can easily connect to Tableau and RShiny. We ultimately built a visualization using RShiny, that enables the user to interactively visualize and garner insights as necessary.

Methodology

Our methodology can be defined with the following steps:

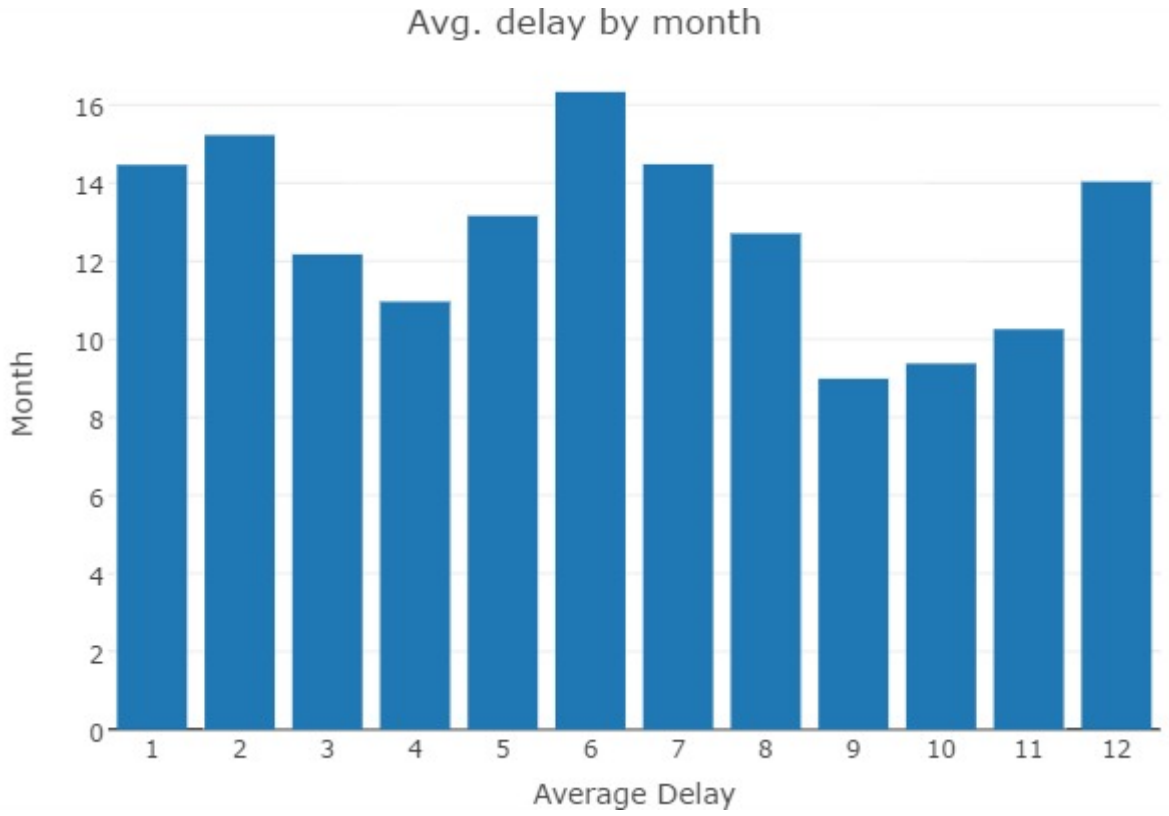
1. Upload data onto AWS S3 bucket
2. Use Hive to process data an create derived variables
3. Using Amazon EC2, create an instance that enables connectivity to RStudio Server
4. Use SparkR and H2O to run predictive modeling algorithms on the data
5. Visualize results using RShiny

Results

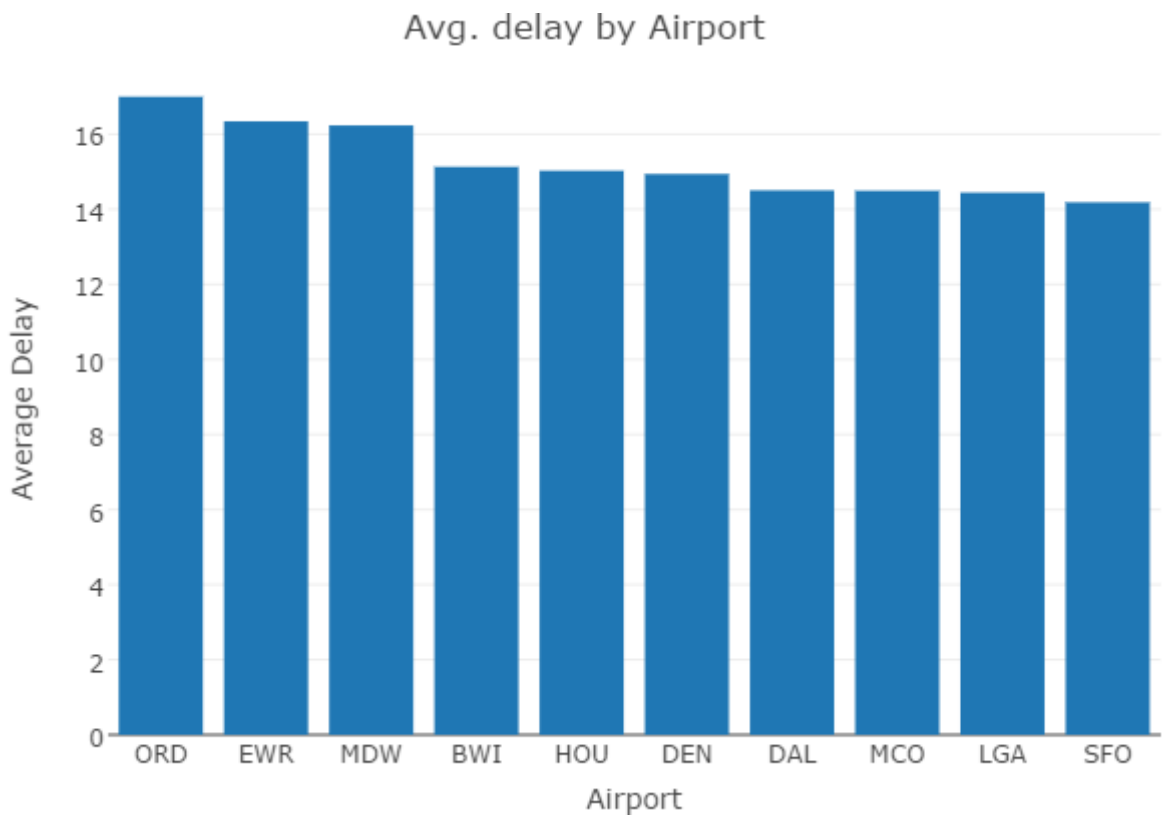
Exploratory Analysis

After performing basic exploratory analysis on our data, we wanted to explore the various factors that could affect how much a flight would get delayed. With the attributes that we had, we decided to look at how the average delay varies across various factors such as month, airport, airline carrier, hour of day etc.

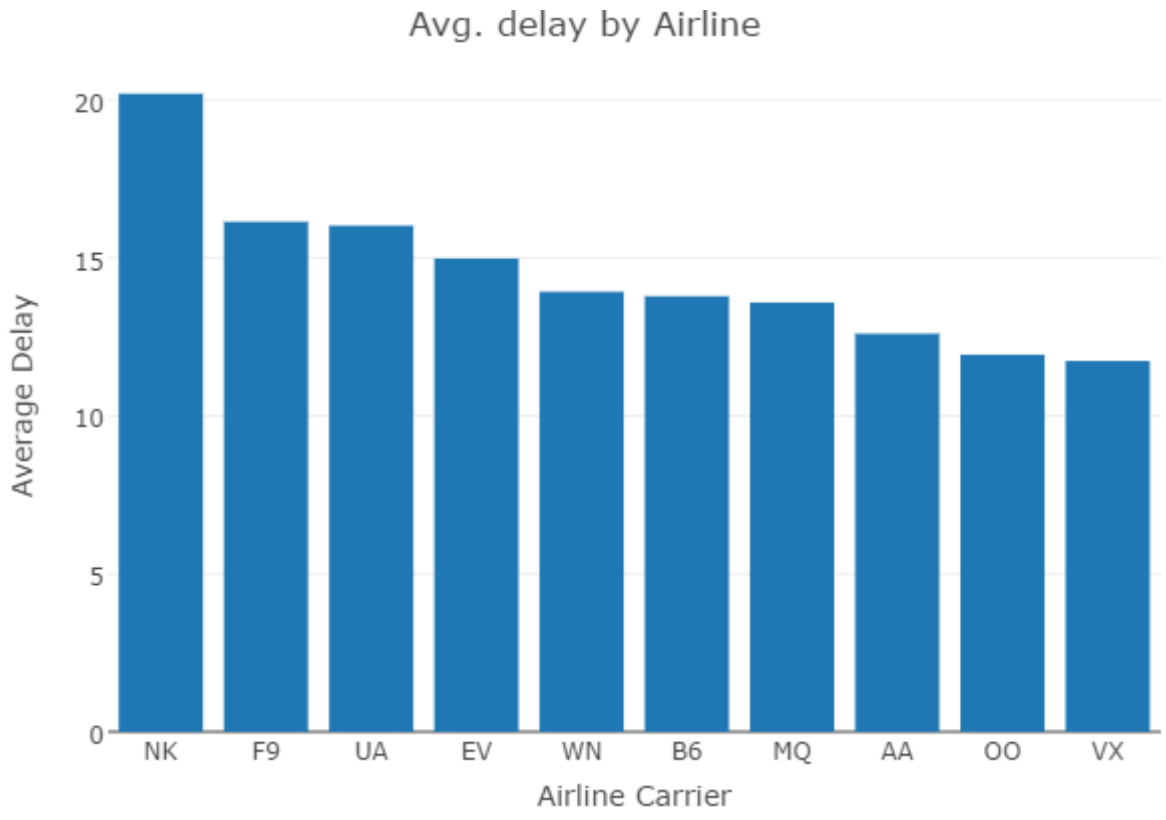
Average delay by month



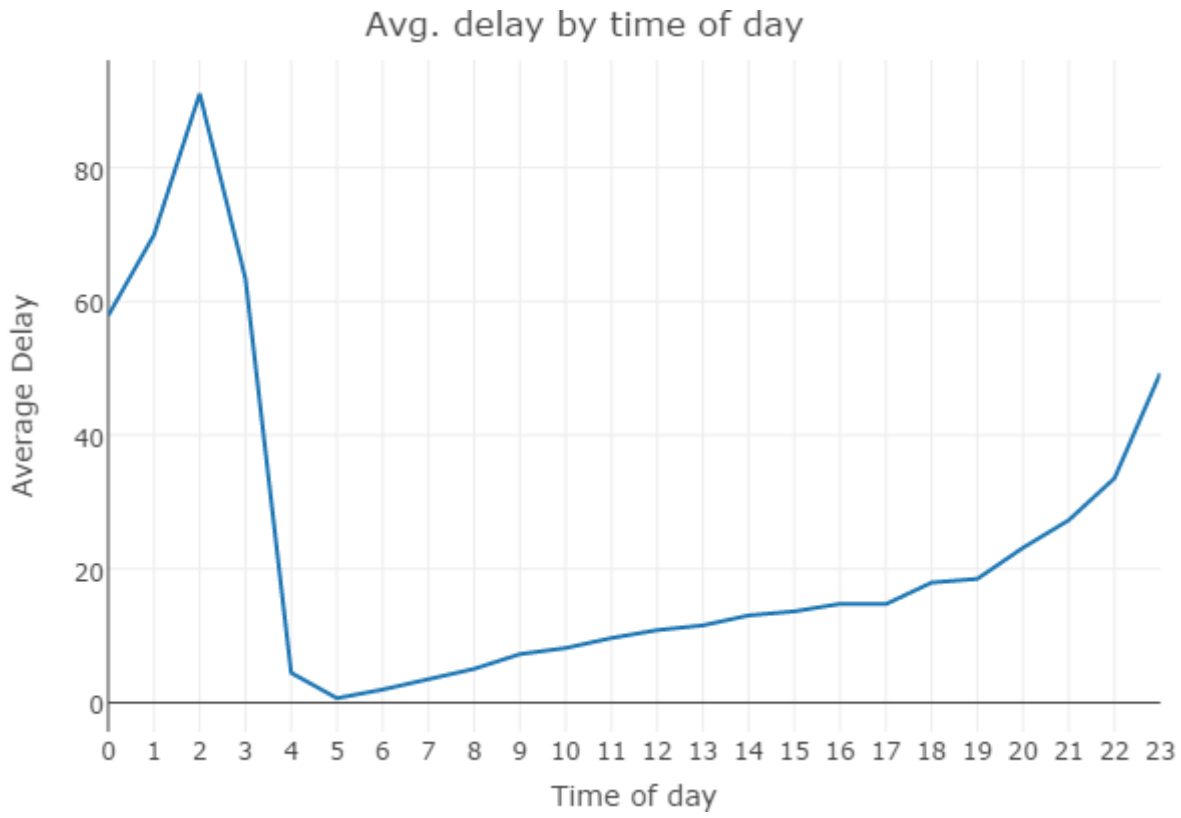
Average delay by airport (top 10)



Average delay by carrier (top 10)



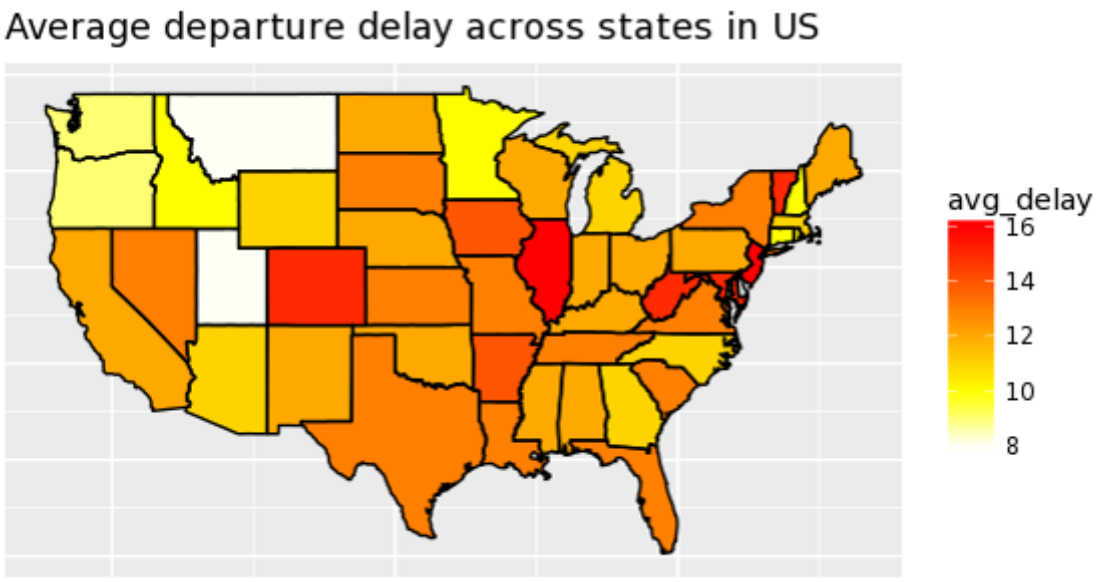
Average delay by hour of day



From the above graphs, we observe the following:

- June, July and December have the highest average delays
- O'Hare, Newark and Chicago mid-west airports have the highest average delays
- Spirit Airlines, Frontier airlines and United Airlines have the highest average delay among airlines
- Delays are greater between 10 AM and 2 PM, possibly because of poor visibility

We also decided to explore states that have the highest average delay in their airports. A heat map as shown below gives us a general overview of this:



Modeling

We used various predictive modeling techniques using H2O to see which of the techniques gives us the best accuracy. Our target variable was a binary variable indicating whether a flight was delayed by 15 minutes or greater or not. Below is an example of the neural network algorithm on our dataset using H2O:

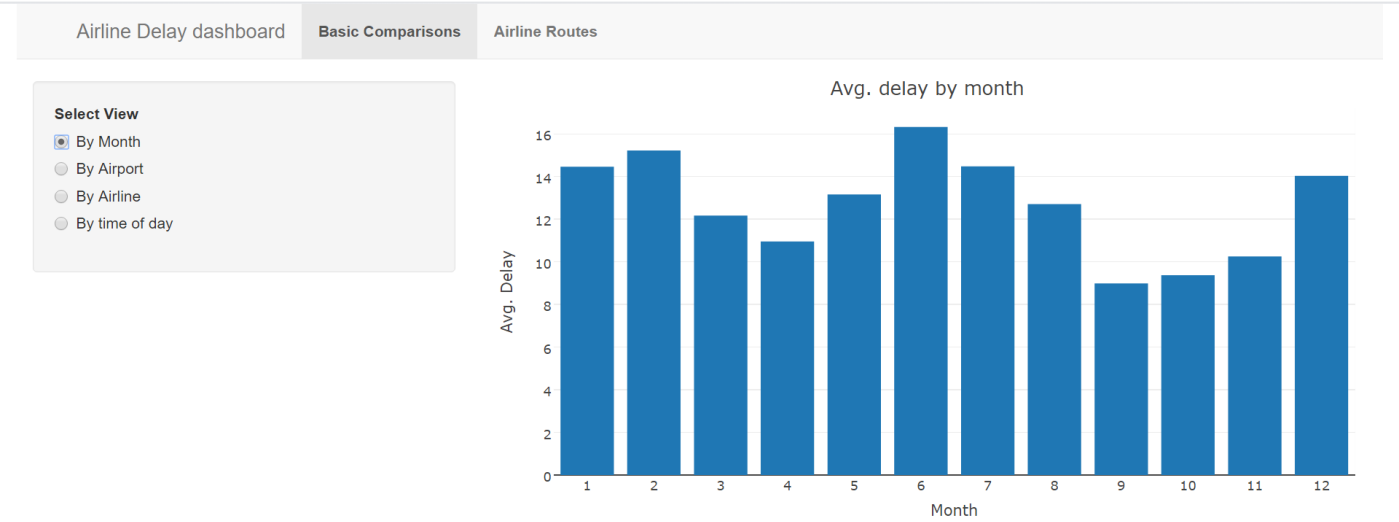
```
fit_nn <- h2o.deeplearning(
  x = c("month",
        "dayofweek",
        "carrier",
        "distance",
        "traffic",
        "dephour_bucket"),
  y = "delay_flag",
  training_frame = training,
  validation_frame = test
)
```

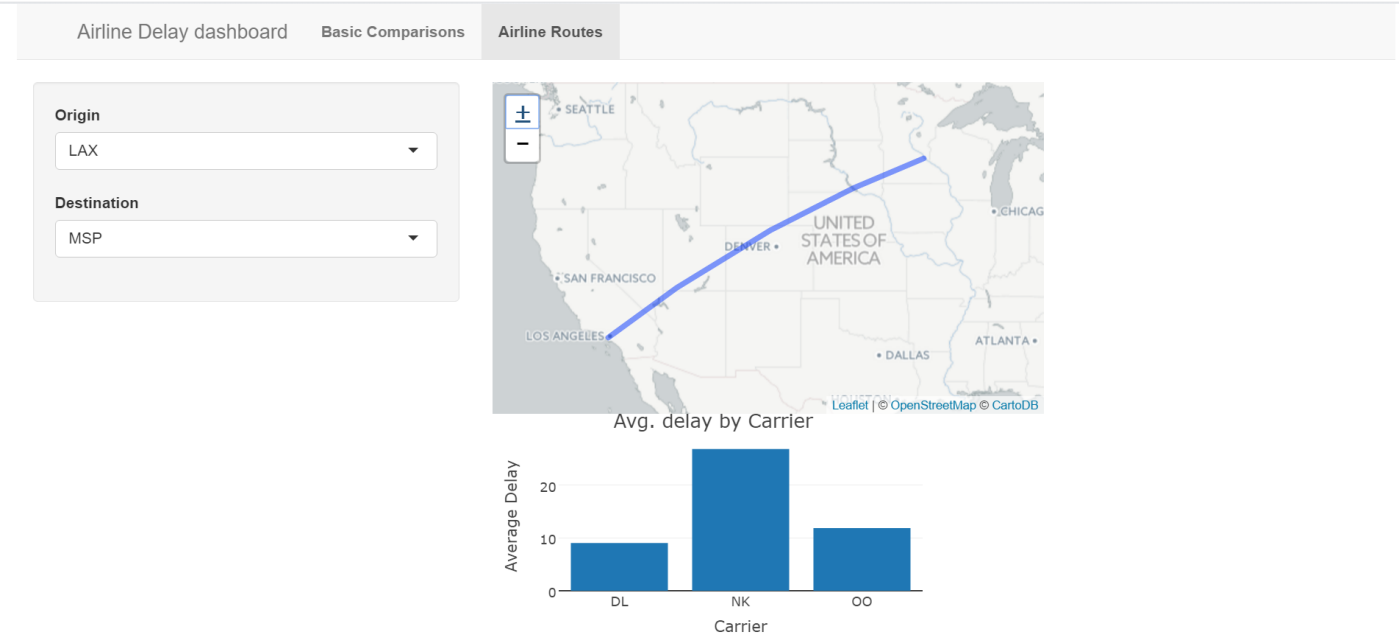
Below, we summarize the results of each of the models by comparing their accuracies:

Model	Accuracy
Logistic Regression	56 %
Naïve Bayes	55 %
Random Forest	66 %
Neural Networks	74 %
Gradient Boost	60 %

Shiny Dashboard

In order to visualize the results in an intuitive manner, we developed a dashboard using RShiny. The dashboard is dynamically connected to the R Server, which preserves the spark and H2O connectivity. The dashboard enables the user to visualize basic plots, and also a flight route plot to understand which carriers perform bad with respect to delays on a prticular route.





Next Steps

- Fine tune model and extract more attributes for more predictive accuracy
- Cluster airports that are similar to identify delay patterns within airport segments
- Stream real time flight data to provide dynamic and accurate predictions

Conclusion

With the volume of data growing exponentially over time, it is imperative that corporations switch to frameworks and platorms that are scalable. With the advent and growth of Hadoop and Spark, and machine learning platforms such as MLLib and H2O, it is essential that companies embrace these technologies openly to garner insights from huge volumes of data and institutionalize data driven decision making.

Appendix

Hive code to process data

```
DROP TABLE IF EXISTS airdata;
CREATE EXTERNAL TABLE airdata (
Year STRING,
Month STRING,
DayofMonth STRING,
DayOfWeek STRING,
FlightDate STRING,
UniqueCarrier STRING,
Carrier STRING,
OriginAirportID STRING,
Origin STRING,
OriginState STRING,
DestAirportID STRING,
Dest STRING,
DestState STRING,
DepTime STRING,
Depdelay STRING,
ArrTime STRING,
ArrDelay STRING,
Cancelled STRING,
CancellationCode STRING,
Distance STRING,
CarrierDelay STRING,
WeatherDelay STRING,
NASDelay STRING,
SecurityDelay STRING,
LateAircraftDelay STRING,
FirstDepTime STRING,
TotalAddGTime STRING,
LongestAddGTime STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3://projectairline/neworgdata';

-- replace quotes and redundancy
drop table if exists cleandata;
create table cleandata as
select
regexp_replace(Year, '\\"', '') as Year,
regexp_replace(Month, '\\"', '') as Month,
regexp_replace(DayofMonth, '\\"', '') as DayofMonth,
regexp_replace(DayOfWeek, '\\"', '') as DayOfWeek,
regexp_replace(FlightDate, '\\"', '') as FlightDate,
regexp_replace(UniqueCarrier, '\\"', '') as UniqueCarrier,
regexp_replace(Carrier, '\\"', '') as Carrier,
regexp_replace(OriginAirportID, '\\"', '') as OriginAirportID,
regexp_replace(Origin, '\\"', '') as Origin,
regexp_replace(OriginState, '\\"', '') as OriginState,
```

```
regexp_replace(DestAirportID, '\\"', '') as DestAirportID,
regexp_replace(Dest, '\\"', '') as Dest,
regexp_replace(DestState, '\\"', '') as DestState,
regexp_replace(DepTime, '\\"', '') as DepTime,
regexp_replace(Depdelay, '\\"', '') as Depdelay,
regexp_replace(ArrTime, '\\"', '') as ArrTime,
regexp_replace(ArrDelay, '\\"', '') as ArrDelay,
regexp_replace(Cancelled, '\\"', '') as Cancelled,
regexp_replace(CancellationCode, '\\"', '') as CancellationCode,
regexp_replace(Distance, '\\"', '') as Distance,
regexp_replace(CarrierDelay, '\\"', '') as CarrierDelay,
regexp_replace(WeatherDelay, '\\"', '') as WeatherDelay,
regexp_replace(NASDelay, '\\"', '') as NASDelay,
regexp_replace(SecurityDelay, '\\"', '') as SecurityDelay,
regexp_replace(LateAircraftDelay, '\\"', '') as LateAircraftDelay,
regexp_replace(FirstDepTime, '\\"', '') as FirstDepTime,
regexp_replace(TotalAddGTime, '\\"', '') as TotalAddGTime,
regexp_replace(LongestAddGTime, '\\"', '') as LongestAddGTime
from airdata;

-- remove column name row
drop table if exists cleandata1;
create table cleandata1 as
select * from cleandata
where lower(year) not like "%year%";


-- generate airport traffic column
drop table if exists cleandata2;
create table cleandata2 as
select *, substring(deptime, 1,2) as dephour, substring(arrtime, 2,2) as arrhour fr
om cleandata1;

-- group by
drop table if exists dephourtable;
create table dephourtable as
select year, month, dayofmonth, dephour, originairportid, count(*) as deptraffic
from cleandata2
group by year, month, dayofmonth, dephour, originairportid;

-- group by2
drop table if exists arrhourtable;
create table arrhourtable as
select year, month, dayofmonth, arrhour, destairportid, count(*) as arrtraffic
from cleandata2
group by year, month, dayofmonth, arrhour, destairportid;

-- merge groupby
```

```
drop table if exists trafficdata;
create table trafficdata as
select d.year, d.month, d.dayofmonth, d.dephour, d.originairportid,
if(d.deptraffic is null,0,d.deptraffic)+if(a.arrtraffic is null,0,a.arrtraffic) as
traffic
from dephourtable d full outer join arrhourtable a
on (d.year = a.year)
and (d.month = a.month)
and (d.dayofmonth = a.dayofmonth)
and (d.dephour = a.arrhour)
and (d.originairportid = a.destairportid);


-- merge
drop table if exists mergetable;
create table mergetable as
SELECT c.Year,
c.Month,
c.DayofMonth,
c.DayOfWeek,
c.FlightDate,
c.UniqueCarrier,
c.Carrier,
c.OriginAirportID,
c.Origin,
c.OriginState,
c.DestAirportID,
c.Dest,
c.DestState,
c.DepTime,
c.Depdelay,
c.ArrTime,
c.ArrDelay,
c.Cancelled,
c.CancellationCode,
c.Distance,
c.CarrierDelay,
c.WeatherDelay,
c.NASDelay,
c.SecurityDelay,
c.LateAircraftDelay,
c.FirstDepTime,
c.TotalAddGTime,
c.LongestAddGTime,
t.traffic,
t.dephour
FROM cleandata2 c join trafficdata t
where (c.year = t.year)
```

```
and (c.month = t.month)
and (c.dayofmonth = t.dayofmonth)
and (c.dephour = t.dephour)
and (c.originairportid = t.originairportid);

-- convert data type
drop table if exists finaldata;
create table finaldata
row format delimited fields terminated by ',' as
SELECT Year,
Month,
DayofMonth,
DayOfWeek,
FlightDate,
UniqueCarrier,
Carrier,
OriginAirportID,
Origin,
OriginState,
DestAirportID,
Dest,
DestState,
DepTime,
cast(Depdelay as bigint),
ArrTime,
cast(Arrtime as bigint)
Cancelled,
cast(Distance as bigint),
CancellationCode,
CarrierDelay,
WeatherDelay,
NASDelay,
SecurityDelay,
LateAircraftDelay,
FirstDepTime,
TotalAddGTime,
LongestAddGTime,
traffic,
dephour
from mergetable;

-- add city
DROP TABLE IF EXISTS city;
CREATE EXTERNAL TABLE city
(OriginAirportID STRING,
OriginCity STRING,
OriginState STRING,
DestAirportID STRING,
```

```
DestCity STRING,
DestState STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3://projectairline/airportcity';

drop table if exists cleancity;
create table cleancity as
select
regexp_replace(OriginAirportID, '\\"', '') as OriginAirportID,
regexp_replace(OriginCity, '\\"', '') as OriginCity,
regexp_replace(OriginState, '\\"', '') as OriginState,
regexp_replace(DestAirportID, '\\"', '') as DestAirportID,
regexp_replace(DestCity, '\\"', '') as DestCity,
regexp_replace(DestState, '\\"', '') as DestState
from city;

DROP TABLE IF EXISTS origincity;
CREATE TABLE origincity as
select distinct OriginAirportID, OriginCity
from cleancity;

DROP TABLE IF EXISTS destcity;
CREATE TABLE destcity as
select distinct DestCity as destairportid, Deststate as destcity
from cleancity;

DROP TABLE IF EXISTS finaldata;
CREATE EXTERNAL TABLE finaldata
(year                    string,
month                   string,
dayofmonth              string,
dayofweek               string,
flightdate              string,
uniquecarrier           string,
carrier                 string,
originairportid         string,
origin                  string,
originstate             string,
destairportid           string,
dest                    string,
deststate               string,
deptime                 string,
depdelay                bigint,
arrtime                 string,
cancelled               bigint,
distance                bigint,
cancellationcode        string,
```

```
carrierdelay            string,
weatherdelay            string,
nasdelay                string,
securitydelay           string,
lateaircraftdelay       string,
firstdeptime            string,
totaladdgtime           string,
longestaddgtime         string,
traffic                 bigint,
dephour                 string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION 's3://projectairline/newoutput';

DROP TABLE IF EXISTS finalcity;
CREATE TABLE finalcity AS
SELECT f.year,
f.month,
f.dayofmonth,
f.dayofweek,
f.flightdate,
f.uniquecarrier,
f.carrier,
f.originairportid,
f.origin,
f.originstate,
f.destairportid,
f.dest,
f.deststate,
f.deptime,
f.depdelay,
f.arrtime,
f.cancelled,
f.distance,
f.cancellationcode,
f.carrierdelay,
f.weatherdelay,
f.nasdelay,
f.securitydelay,
f.lateaircraftdelay,
f.firstdeptime,
f.totaladdgtime,
f.longestaddgtime,
f.traffic,
f.dephour,
o.OriginCity
FROM finaldata f JOIN origincity o
where o.originairportid = f.originairportid;
```

```
DROP TABLE IF EXISTS finalcity2;
CREATE TABLE finalcity2
row format delimited fields terminated by ',' as
SELECT f.year,
f.month,
f.dayofmonth,
f.dayofweek,
f.flightdate,
f.uniquecarrier,
f.carrier,
f.originairportid,
f.origin,
f.originstate,
f.destairportid,
f.dest,
f.deststate,
f.deptime,
f.depdelay,
f.arrtime,
f.cancelled,
f.distance,
f.cancellationcode,
f.carrierdelay,
f.weatherdelay,
f.nasdelay,
f.securitydelay,
f.lateaircraftdelay,
f.firstdeptime,
f.totaladdgtime,
f.longestaddgtime,
f.traffic,
f.dephour,
f.OriginCity,
d.destcity
FROM finalcity f JOIN destcity d
where d.destairportid = f.destairportid;
```

RShiny App for exploratory analysis (including spark connection and H2O initiation)

```
library(shiny)
library(h2o)
library(sparklyr)
library(rsparkling)
library(dplyr)
library(data.table)
library(dtplyr)
library(plotly)
library(shinythemes)
library(leaflet)
library(geosphere)

h2o.init(nthreads = -1)
spark_install(version = "2.0.0")


conf <- spark_config()
conf[["spark.sql.warehouse.dir"]] <- "/dev/shm"

options(rsparkling.sparklingwater.version = '1.6.8')
sc <- spark_connect(master = "local",
                    spark_home = Sys.getenv("SPARK_HOME"),
                    config = conf)

airline_data <- fread("../finalairline.csv", header = F, stringsAsFactors = F,
                      col.names = c('year','month','dayofmonth','dayofweek','flight
date','uniquecarrier','carrier','originairportid','origin','originstate','destairpo
rtid','dest','deststate','deptime','depdelay','arrtime','cancelled','distance','can
cellationcode','carrierdelay','weatherdelay','nasdelay','securitydelay','lateaircra
ftdelay','firstdeptime','totaladdgtime','longestaddgtime','traffic','dephour'))

air <- read.csv('https://raw.githubusercontent.com/plotly/datasets/master/2011_febr
uary_us_airport_traffic.csv')


times <- substring(paste0("00",0:23),nchar(paste0("00",0:23)) - 1, nchar(paste0("0
0",0:23)))

FullPage <- navbarPage(
  title = "Airline Delay dashboard",
  tabPanel(
    strong("Basic Comparisons"),
    sidebarLayout(
      position = "left",
      sidebarPanel(
        radioButtons("View",
                     label = "Select View",
                     choices = list('By Month',
```

```
                                    'By Airport',
                                    'By Airline',
                                    'By time of day'
                        )
        )
        #,actionButton("action1", label = "Run Model")
      ),
      mainPanel(
        plotlyOutput("model")
      )
    )
  ),
  tabPanel(
    strong("Airline Routes"),
    sidebarLayout(
      position = 'left',
      sidebarPanel(
        selectInput('origin',
                    label = 'Origin',
                    choices = unique(air$iata),
                    selected = 'LAX'),

        selectInput('dest',
                    label = 'Destination',
                    choices = unique(air$iata),
                    selected = 'JFK')#,

              ),
      mainPanel(leafletOutput('airline_perf',width = 500,height = 300),
                plotlyOutput('airplot',width = 400,height = 200))
    )
  )
)


# Define UI for application that draws a histogram
ui <- shinyUI(fluidPage(#theme = "darkly.css",
  #theme = shinytheme("spacelab"),
  title = "Carlson Wagonlit Forecasting Tool",
  FullPage))

server <- shinyServer(
  function(input, output, session) {

    spark_model <- reactive({

      if(input$View == 'By Month') {
```

```
        airlinedata_grouped <- airline_data %>%
          mutate(act_delay = ifelse(as.numeric(depdelay) > 0,
                                    as.numeric(depdelay),0)) %>%
          group_by(month) %>%
          summarize(avg_delay = mean(act_delay, na.rm = T)) %>%
          arrange(month) %>%
          mutate(month_name = as.factor(month.abb[month]))

        #levels(airlinedata_grouped$month_name) = month.abb[1:12]

        p <- plot_ly(airlinedata_grouped,
                     x = ~month,
                     y = ~avg_delay,
                     type = 'bar') %>%
          layout(title = 'Avg. delay by month',
                 xaxis = list(title = 'Month', dtick = 1),
                 yaxis = list(title = 'Avg. Delay'))

      } else if(input$View == 'By Airport') {

        airlinedata_grouped <- airline_data %>%
          mutate(act_delay = ifelse(as.numeric(depdelay) > 0,
                                    as.numeric(depdelay),0)) %>%
          group_by(origin) %>%
          summarize(avg_delay = mean(act_delay, na.rm = T),
                    numrows = n()) %>%
          filter(numrows > 100000) %>%
          arrange(-avg_delay)

        airlinedata_grouped <- head(airlinedata_grouped,10)
        airports <- airlinedata_grouped$origin
        airlinedata_grouped$origin <- factor(airlinedata_grouped$origin, levels = u
nique(airlinedata_grouped$origin)[order(airlinedata_grouped$avg_delay, decreasing =
 TRUE)])

        levels(airlinedata_grouped$origin) <- airports

        p <- plot_ly(airlinedata_grouped,
                     x = ~origin,
                     y = ~avg_delay,
                     type = 'bar') %>%
          layout(title = 'Avg. delay by Airport',
                 xaxis = list(title = 'Airport'),
                 yaxis = list(title = 'Average Delay'))

      } else if(input$View == 'By Airline') {

        airlinedata_grouped <- airline_data %>%
```

```
          mutate(act_delay = ifelse(as.numeric(depdelay) > 0,
                                    as.numeric(depdelay),0)) %>%
          group_by(carrier) %>%
          summarize(avg_delay = mean(act_delay, na.rm = T),
                    numrows = n()) %>%
          filter(numrows > 1000) %>%
          arrange(-avg_delay)

        airlinedata_grouped <- head(airlinedata_grouped,10)
        airports <- airlinedata_grouped$carrier
        airlinedata_grouped$carrier <- factor(airlinedata_grouped$carrier, levels =
 unique(airlinedata_grouped$carrier)[order(airlinedata_grouped$avg_delay, decreasin
g = TRUE)])

        #levels(airlinedata_grouped$carrier) <- carriers

        p <- plot_ly(airlinedata_grouped,
                     x = ~carrier,
                     y = ~avg_delay,
                     type = 'bar') %>%
          layout(title = 'Avg. delay by Airline',
                 xaxis = list(title = 'Airline Carrier'),
                 yaxis = list(title = 'Average Delay'))

      } else {

        airlinedata_grouped <- airline_data %>%
          mutate(act_delay = ifelse(as.numeric(depdelay) > 0,
                                    as.numeric(depdelay),0),
                 dephour = ifelse(as.integer(dephour) == 24,0,as.integer(dephour)))
 %>%
          group_by(dephour) %>%
          summarize(avg_delay = mean(act_delay, na.rm = T)) %>%
          arrange(dephour)

        airlinedata_grouped <- airlinedata_grouped[complete.cases(airlinedata_group
ed),]


        p <- plot_ly(airlinedata_grouped,
                     x = ~as.numeric(dephour),
                     y = ~avg_delay,type = 'scatter', mode = 'lines') %>%
          layout(title = 'Avg. delay by time of day',
                 xaxis = list(title = 'Time of day', dtick = 1),
                 yaxis = list(title = 'Average Delay'))

      }
```

```
  p

})

origin <- reactive({
  req(input$origin)
  filter(air, iata == input$origin)
})

# Identify destination lat and log
dest <- reactive({
  req(input$dest)
  filter(air, iata == input$dest)
})

by_airport <- reactive({

  airline_data_grouped <- airline_data %>%
    filter(origin == input$origin &
             dest == input$dest) %>%
    mutate(act_delay = ifelse(as.numeric(depdelay) > 0,
                              as.numeric(depdelay),0),
           dephour = ifelse(as.integer(dephour) == 24,0,as.integer(dephour))) %
>%
    group_by(carrier) %>%
    summarize(avg_delay = mean(act_delay, na.rm = T))

  p <- plot_ly(airline_data_grouped,
               x = ~carrier,
               y = ~avg_delay,
               type = 'bar') %>%
    layout(title = 'Avg. delay by Carrier',
           xaxis = list(title = 'Carrier'),
           yaxis = list(title = 'Average Delay'))

  p

})



output$airplot <- renderPlotly({

  by_airport()

})

output$model <- renderPlotly({
```

```
      spark_model()

    })

    output$airline_perf <- renderLeaflet({
      gcIntermediate(
        select(origin(), long, lat),
        select(dest(), long, lat),
        n=100, addStartEnd=TRUE, sp=TRUE
      ) %>%
        leaflet() %>%
        addProviderTiles("CartoDB.Positron") %>%
        addPolylines()


    })



  })

# Run the application
shinyApp(ui = ui, server = server)
```

Models using H2O

```
###Naive Bayes model
partitions <- airline_data_final %>%
  sdf_partition(training = 0.8, test = 0.2, seed = 42)

training <- as.h2o(partitions$training)
test <- as.h2o(partitions$test)

training$month <- as.factor(training$month)
training$dephour <- as.factor(training$dephour)
training$dayofweek <- as.factor(training$dayofweek)
training$carrier <- as.factor(training$carrier)
training$dephour_bucket <- as.factor(training$dephour_bucket)

trainx = c("month","dayofweek","carrier", "distance", "traffic","dephour_bucket")
training$delay_flag = as.factor(training$delay_flag)


nb_fit <- h2o.naiveBayes(trainx, 'delay_flag', training, laplace = 0)

summary(nb_fit)

##Logistic Regression
logit_fit <- h2o.glm(
  x = c("month",
        "dayofweek",
        "carrier",
        "distance",
        "traffic",
        "dephour_bucket"),
  y = "delay_flag",
  training_frame = training,
  lambda_search = TRUE,
  standardize = TRUE,
  family = 'binomial',
)

##Neural Networks
fit_nn <- h2o.deeplearning(
  x = c("month",
        "dayofweek",
        "carrier",
        "distance",
        "traffic",
        "dephour_bucket"),
  y = "delay_flag",
  training_frame = training,
  validation_frame = test
)
```

```
##Random forest
fit_rf <- h2o.randomForest(

    training_frame = training,
    validation_frame = test,
    x = trainx,
    y = 'delay_flag',
    model_id = "rf_covType_v1",
    ntrees = 200,
    stopping_rounds = 2,
    # score_each_iteration = TRUE,
    seed = 1000000
)

##Gradient boosted model
fit_gbm <- h2o.gbm(

    training_frame = training,
    validation_frame = test,
    x = trainx,
    y = 'delay_flag',
    model_id = "gbm_covType1",
    seed = 2000000
)
```