My Comprehensive Evaluation

A Comprehensive Evaluation Report
Presented to
The Statistics Faculty
Amherst College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts
in
Statistics

Azka Javaid

Feburary 2017

Acknowledgements

| I would first like to thank the Amherst College Department of Mathematics and Statistics and Amherst College Department of IT for facilitating this project. I would particularly like to thank Professor Amy Wagaman and Professor Shu-Min Liao for reviewing the project as well as offering suggestions. Additionally I would like to thank Professor Susan Wang and Professor Eunice Kim for their help and support in the Statistics Program.

I would also like to extend a big thanks to the Amherst College Department of IT. This project would not have been possible if it wasn't for Aaron Coburn's continual assistance with setting up the Hadoop cluster and for loading the Flights dataset from 2008-2016. Additionally, I would like to thank Andy Anderson for his continued support with the R server.

I like to thank my classmates from Advanced Data Analysis, particularly Stephany-Flores Ramos, Jordan Browning and Levi Lee for their candid support. I wouldn't have the necessary data wrangling and analytical skills if it wasn't for the cheery class mood created through group work with you all. I would also like to give a special acknowledgement to Muling Si and Levi Lee and their group project presentation from Advanced Data Analysis on mapping course networks for inspiring the flights network in my comps Shiny app. Additionally I would like to thank Caleb Ki. Thanks for all help in navigating Google BigQuery and Shiny and overall for being a great project partner for Advanced Data Analysis! I would also like to thank my roommate, Jane Pierre, for being a keen audience for my shiny apps and an engaging conversationalist for the statistical chit-chats.

Lastly I would like to offer a huge thank you to Professor Nicholas Horton for all his support with the Hadoop and H2O machine learning platform. Thanks for entrenching the importance of both statistical analysis and computing, git programming, collaborative learning and for all encouragement and support with both statistics and computer science majors!

Table of Contents

List of Tables

List of Figures

Abstract

| H2O platform was used to perform logistic regression and deep learning modeling. Three different models were performed. A simple logitistic regression was used to predict departure delay greater than 30 minutes against year, arrival delay, carrier, air time, distance, week and season. A subsequent logistic regression model was built to study influence of weather in predicting the occurrence of departure delay greater than 30 minutes against season, month, week, weekend, day, hour, arrival delay, distance and air time. Weather predictors included temperature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility. Lastly a deep learning model was built to study the influence of variables year, month, arrival delay, carrier, distance, hour, week, weekend and season in predicting departure delay greater than 90 minutes. Additionally grid and random search along with checkpoint models were developed to optimize for hyperparameters.

Arrival delay appeared to be the most important predictor of departure delay which is intuitive suggesting that flights that experienced arrival delay also experienced prior departure delay with a high likelihood. This result also implies that flights were possibly not able to recover from the initial departure delay leading to subsequent arrival delay. Additionally distance and amount of air time were important predictors for departure delay. Delay also appeared to be higher during the early morning hours and on weekends. The results also point to an increased trend in recent years like 2015 in comparison to years like 2008 and 2009. Additionally weather does not appear to be very important in predicting departure delay. Future studies can be performed to better inspect the influence of weather over departure delays. The influence of additional possible confounders like previous late flights on departure delays can be discussed.

Chapter 1

Introduction

1.1 Motivation

Airline-caused delays (late arrival of 15 minutes) totaled 20.2 million minutes in 2015. About 14.3 million minutes of delay was caused by weather, congested airports and air traffic system complications. Severe weather and security concerns resulted in delays of about 17.5 million minutes while about 25 million minutes of delay was allotted to undetermined reasons like a previously delayed flight. In total, this resulted in about 77 million minutes of delay which converts to about 1 million and 283 thousand hours of delay in 2015 (Levin & Sasso, 2016).

The purpose of this study was manifold. The intent was to explain the occurrence of departure delays from 2008 to 2016 which resulted in about 1 million and 283 thousand hours of lost time in 2015. Additionally, the interest was in exploring big data platforms like Hadoop and Apache Spark. The main intent of this work was the exploration of the deep learning platform, H2O, and its application on predicting departure delays.

Chapter 2

Sparkling Infrastructure

2.1 H2O

H2O is a big-data machine learning and predictive analytics platform, reputable for its fast and scalable deep learning capabilities. These include supervised and unsupervised learning algorithms like neural networks, tree ensembling, generalized linear regression models and k-means clustering. Additionally H2O provides deep learning capabilities through algorithms like perceptrons and feed forward neural networks. In its essence, H2O is a Java Virtual Machine (JVM), which is an abstract computing environment for running a Java instance. JVM provides a close and secure environment for running Java applications (Rickert, 2014). H2O clients consequently have a remote connection to the data held on the H2O clusters as shown by Figure 2.1 (Cook, 2016). H2O's contained environment also makes it optimal for performing distributed and parallel machine learning computations simultaneously on clusters.

2.1.1 H2O Installation Process

H2O installation in R entails installation of Spark 1.6 and H2O version 3.10.06 since this version of H2O integrated with rsparkling. H2O installation involved removal of any previous H2O versions followed by download of H2O dependency packages and lastly installation and initialization of R H2O packages. H2O package download link can be obtained at (`http://h2o-release.s3.amazonaws.com/h2o/rel-turing/6/index.html#R`)[1]. Rsparkling package is downloaded using devtools.

Before using H2O's machine learning algorithms, the same version of Spark and Sparkling Water needs to be installed in R. Desktop version of Spark can be downloaded at (`http://spark.apache.org/downloads.html`)[2]. Desktop version of Sparkling Water can be downloaded at (`http://h2o-release.s3.amazonaws.com/sparkling-water/rel-1.6/8/index.html`)[3]. H2O also necessitates initialization of a local Spark connection, which can be instantiated with the spark_connect function. In R, H2O connection was established with the local cluster. In comparison, for Hadoop platform,

[1](“Download h2o 3.10.0.6,” n.d.)
[2](“Download apache spark,” n.d.)
[3](“Download sparkling water 1.6.8,” n.d.)

H2O connection can be established to the YARN-client discussed in the Hadoop YARN section.

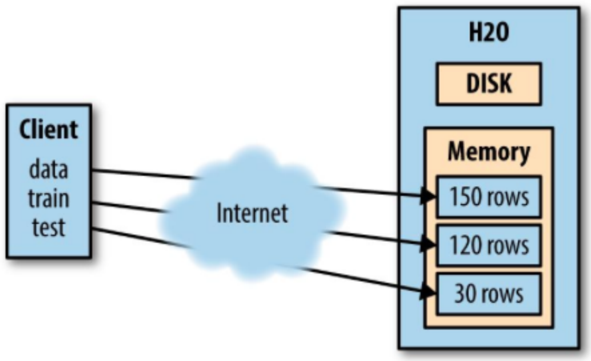


Figure 2.1: Client and H2O Cluster

2.2 Apache Spark

Spark is a big-data platform that provides fast in-memory distributed processing. This contrasts with Hadoop, which employs the MapReduce processing platform and necessitates data writing to an external disk through the Hadoop Distributed File System (HDFS) (Borthakur, n.d.). Spark uses the Resilient Distributed Datasets (RDD) data structure, which divides the dataset in logical partitions each of which can be processed on separate nodes within clusters. This RDD structure obviates the need to write data to an external storage system thus providing faster in-memory processing ("Spark programming guide," n.d.). Moreover Apache Spark is an in-memory data processing tool that hosts the Spark platform on Apache Hadoop YARN providing a collective and shared access to a dataset.

In R, the sparklyr package provides an R interface for Apache Spark. Besides a dplyr background, this facilitates the use of Spark's distributed machine learning library.

2.3 Sparkling Water

Sparkling water combines the machine learning capabilities of H2O with the in-memory distributed, fast computation of the Spark platform. Tachyon, which is an in-memory distributed file system, facilitates exchange of data between Spark and H2O (Ambati, n.d.). The rsparkling package in R provides access to H2O's machine learning routines within the Spark platform accessible over R. Spark data frames can be converted to H2O frames thus facilitating machine learning algorithms.

2.4 RSparkling

The rsparkling package facilitates data transfer between Spark and H2O dataframes. Rsparkling also allows access to Sparkling Water spark package's machine learning algorithms ("Sparkling water (h20) machine learning," n.d.).

2.5 Hadoop YARN

Apache Hadoop YARN includes separate resource management and job scheduling infrastructures in collective resource manager and individual node manager through a master-slave hierarchy (as shown by Figure 2.2("Spark programming guide," n.d.)). The per-application based application masters negotiate resources with the resource manager and execute and monitor tasks through a collaboration with the node managers. The resource managers additionally contain a scheduler that schedules jobs based on resource requirements. Individual node managers are responsible for launching application containers and examining resource usage (like memory and disk consumption). These updates are then reported back to the resource manager. Per-node application master negotiate resource containers with scheduler (Murthy, 2012).

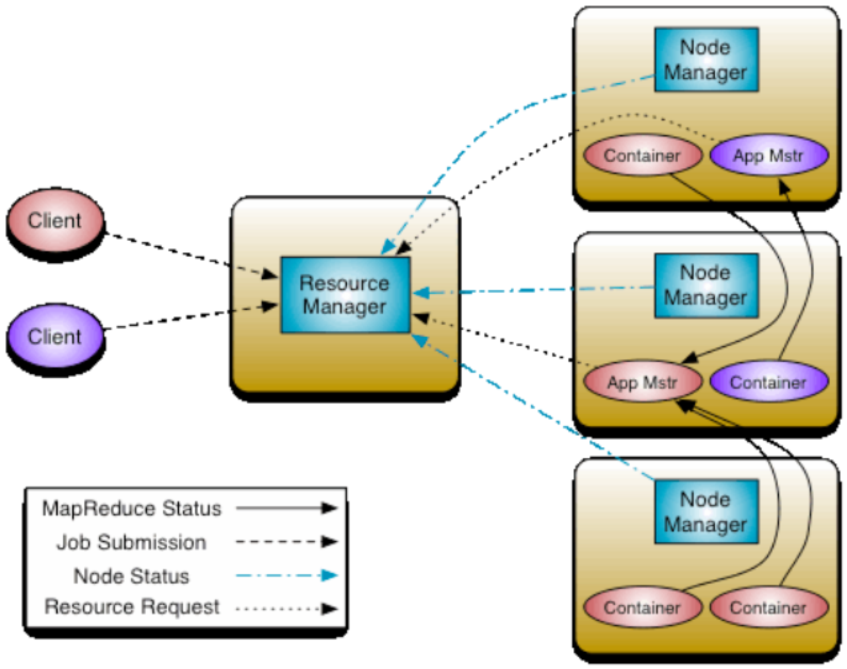


Figure 2.2: Hadoop YARN architecture

2.6 Additional Resources

- H2O Installation guide - `http://h2o-release.s3.amazonaws.com/h2o/rel-turing/6/index.html#R`
- H2O Documentation - `http://h2o-release.s3.amazonaws.com/h2o/rel-lambert/5/docs-website/index.html`
- Sparkling Water Installation - `http://www.h2o.ai/download/sparkling-water/`
- Sparkling Water Overview - `http://spark.rstudio.com/h2o.html`
- Sparklyr Overview and Installation - `http://spark.rstudio.com`

Chapter 3

Shiny Explorations

3.1 Introduction

For this analysis, I used the Flights dataset from the United States Department of Transportation Bureau of Transportation Statistics. Data was set up in the YARN-client cluster in the Hadoop server from years 2008-2016. Initially it contained variables like year, month and day of the trip, departure delay, arrival delay, carrier, tailnumber, distance covered, flight number, flight origin, destination and scheduled flight time. Additional predictors were created to better gauge the departure delay. These variables included day of week, season and weekend status and hour of flight delay.

3.2 Web Scraping

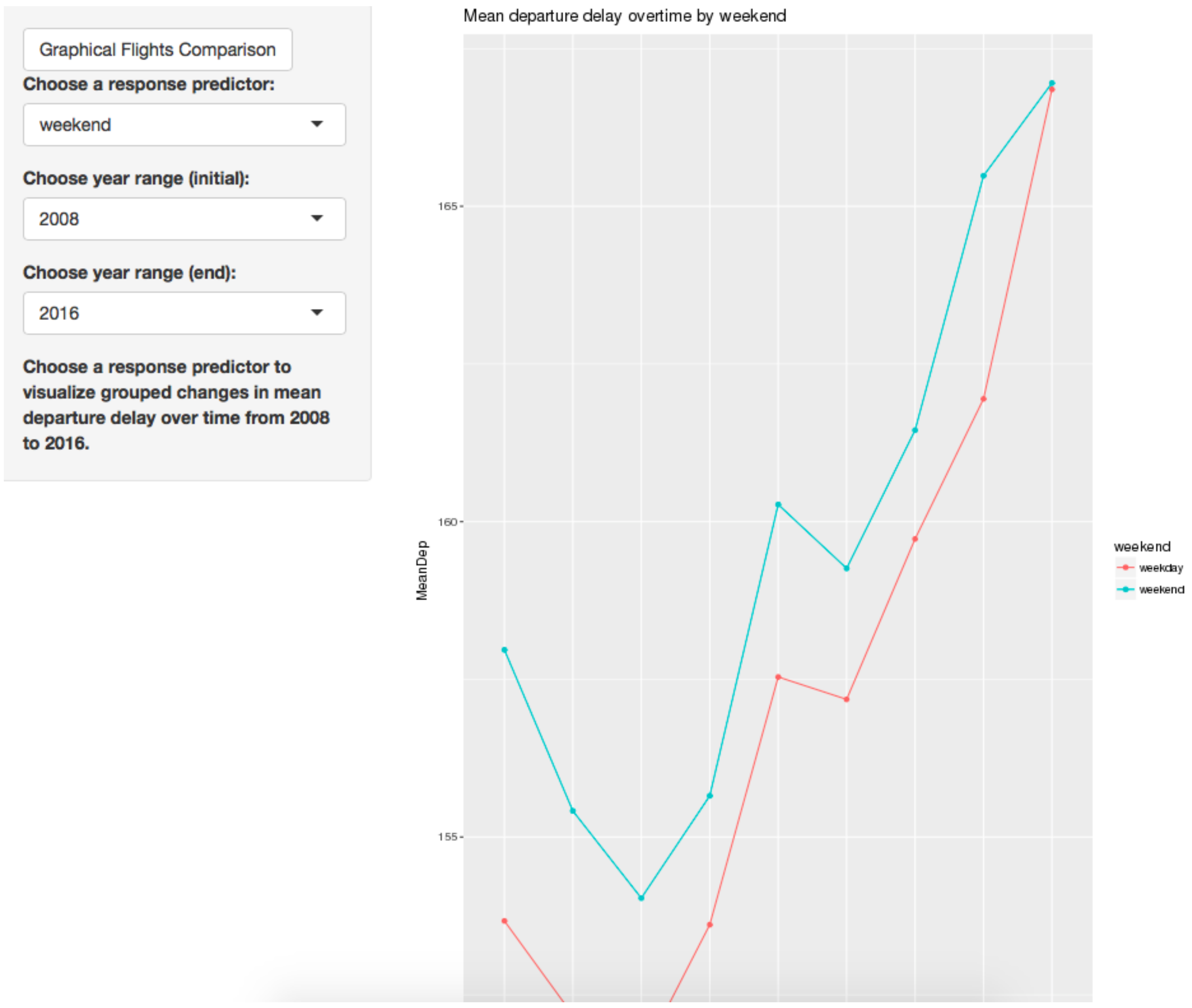
Since carrier, plane origin and destination airport information was provided as two and three letter code names, following the guidelines set by the International Air Transport Association (IATA), additional data was scraped from web to include the origin and destination airport information as well as the carrier and state names. This data was then merged with the flights dataset. Data scraping was performed using rvest package and the SelectorGadget tool, a Chrome extension that allows for easy CSS webpage selection (see Appendix 2 for complete scraping code).

3.3 Shiny App

Shiny App was used for initial exploratory analysis. Graphical, tabular and weather analysis was performed. In addition, networks were constructed to visualize number of flights and departure delays between airports. Images from the shiny app are shown below each commentary. The app itself can be accessed at `https://r.amherst.edu/apps/ajavaid17/Comps2017Flights/FlightsExpo/` (wait time of about half a minute and ignore red ‘ errors).

3.3.1 Graphical Analysis

Graphical shiny analysis was performed on flights with departure delays greater than 90 minutes. About 1590467 flights satistied this criterion and were used for analysis. Departure delays were analyzed graphically by carrier, month, hour, week and weekend status predictors. Mean departure delay was computed by averaging the departure delay in minutes over the user specified year and carrier, month, hour, week or weekend status predictors. The package ggplot2 was used for graphical analysis and reactivity was employed to add dynamicity to the graphs. An example shiny output of changes in mean departure delay from 2008 to 2016 grouped by weekend is shown by Figure 3.1.



Graphical Flights Comparison

Choose a response predictor: weekend

Choose year range (initial): 2008

Choose year range (end): 2016

Choose a response predictor to visualize grouped changes in mean departure delay over time from 2008 to 2016.

Figure 3.1: Graphical Analysis

Departure delay by carrier:

According to the carrier graphical analysis, Hawaiian airlines had a consistently higher delay in comparison to other airlines from 2008-2016. Hawaiian also experienced the most fluctuations in the average departure delays. Southwest and US Air had lowest delays. The overall trend though points to slightly increased departure delays from 2008 to 2016.

Departure delay by month/seasons:

Graphically, differences in departure delay by month appeared to be indistinguishable. To better differentiate the delays by month, season variable was created. Delays for summer appear to be slightly higher than the delays for fall, spring and winter from years 2008 to 2012. The general trend points to an increased delay from 2008 to 2016.

Departure delay by hour/time of day status:

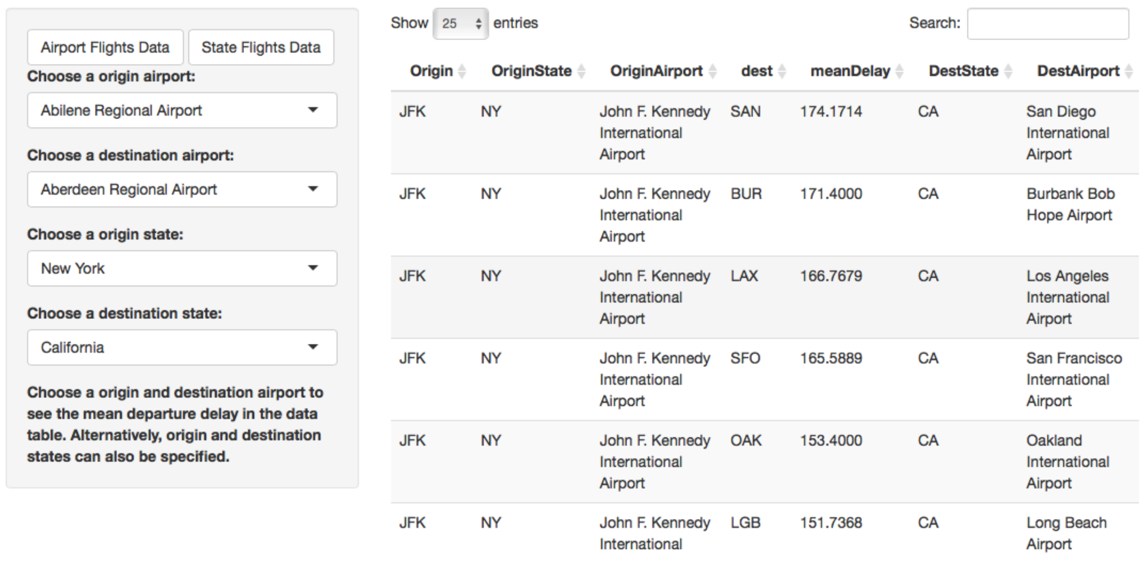
Hours 0-4 (12 am to 4 am) appeared to be most unpredictable in mean departure delay. In comparison, mean departure delay seemed to stablize after 2 pm continuing in the evening hours. Though there was a slight delay in the evening hours, it did not appear to be as problematic as the early morning hour delays.

Departure delay by week/weekend:

Differences in mean departure delay appeared to be indistinguishable by week status. In comparison, there appeared to be higher delays on weekends than weekdays.

3.3.2 Tabular Analysis

The tabular analysis showed the mean departure delay for all flights for the specified origin and destination states. In addition, the panel showed departure delays for all flights for the user selected origin and destination airports. Data Table was used which provided searching functionalities to the app. An output of flights from New York to California from the shiny app for the tabular analysis section is shown by Figure 3.2.



Choose a origin airport: Abilene Regional Airport

Choose a destination airport: Aberdeen Regional Airport

Choose a origin state: New York

Choose a destination state: California

Choose a origin and destination airport to see the mean departure delay in the data table. Alternatively, origin and destination states can also be specified.

Origin	OriginState	OriginAirport	dest	meanDelay	DestState	DestAirport
JFK	NY	John F. Kennedy International Airport	SAN	174.1714	CA	San Diego International Airport
JFK	NY	John F. Kennedy International Airport	BUR	171.4000	CA	Burbank Bob Hope Airport
JFK	NY	John F. Kennedy International Airport	LAX	166.7679	CA	Los Angeles International Airport
JFK	NY	John F. Kennedy International Airport	SFO	165.5889	CA	San Francisco International Airport
JFK	NY	John F. Kennedy International Airport	OAK	153.4000	CA	Oakland International Airport
JFK	NY	John F. Kennedy International	LGB	151.7368	CA	Long Beach Airport

Figure 3.2: Tabular Analysis

3.3.3 Graphical Weather Analysis

In addition to the general analysis for all flights with departure delays greater than 90 minutes from 2008 to 2016, analysis was performed to gauge the effect of weather on total delay, quantified by sum of departure and arrival delays for LaGuardia, John F. Kennedy and Newark Liberty International Airport. Weather analysis was only performed for 2013 since data for this year was readily available in the nycflights13 package. In comparison, weather analysis from 2008 to 2016 would have entailed possible pipeline constructions to fetch data from the National Climatic Data Center (NOAA) by the timestamp specified by the flights data in the Hadoop clusters, a project pursued in subsequent work. The weather data from the nycflights13 package contained about 135059 observations. Effect of weather metrics like temperature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility was analyzed on the total delay over measures like week, weekend, month and seasons. An output from the shiny app is shown in Figure 3.3 depicting changes in total delay by temperature and weekend status.

Total delay by week and weather:

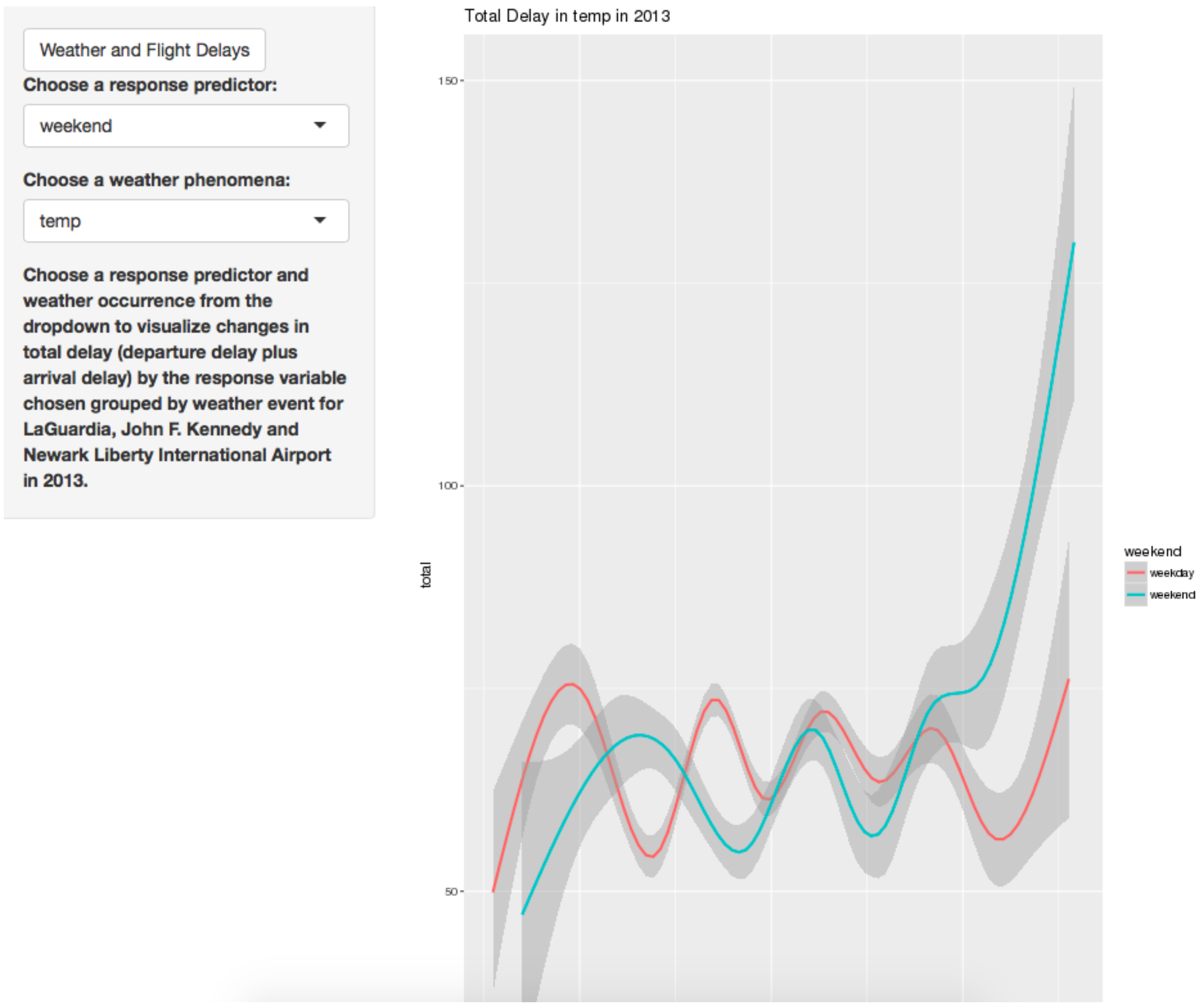
Total delay did not show considerable changes by temperature over the weeks besides a rise in departure delay on Friday with low temperature and a rise in departure delay on Sunday with increase in temperature. Similarly there did not appear to be a visible pattern in total delay by weather patterns over day of week. A general trend pointed towards an increase in total delay with visibility and a decrease in total delay with increased pressure. Increase in humidity appeared to increase departure delay. While Saturday experienced a decrease in total delay with an increase in humidity, this decrease was not necessarily particularly distinguished to be commented on.

Total delay by weekend and weather:

There appeared to be a higher increase in total delay with an increase in temperature on a weekend than a weekday. This higher increase in total delay was also visible with an increase in dewpoint and precipitation on weekends than weekdays. In comparison, increase in total delay on a weekday than weekends with an increase in humidity is apparent. Decrease in total delay with an increase in pressure and visibility was apparent for both weekdays and weekends. For low temperatures, dewpoint, precipitation, humidity, pressure and visibility, differences in weekday and weekends total delays do not appear to be distinguishable.

Total delay by month and weather:

December experienced higher total delays with an increase in temperature in comparison to other months. Differences in total delay with respect to humidity appeared to be indistinguishable by month. Increase in humidity seemed to produce a higher increase in total delay for March, February and December in comparison to other months. There did not appear to be considerable changes in total delays by pressure


Weather and Flight Delays

Choose a response predictor:

weekend

Choose a weather phenomena:

temp

Choose a response predictor and weather occurrence from the dropdown to visualize changes in total delay (departure delay plus arrival delay) by the response variable chosen grouped by weather event for LaGuardia, John F. Kennedy and Newark Liberty International Airport in 2013.

Figure 3.3: Weather Analysis

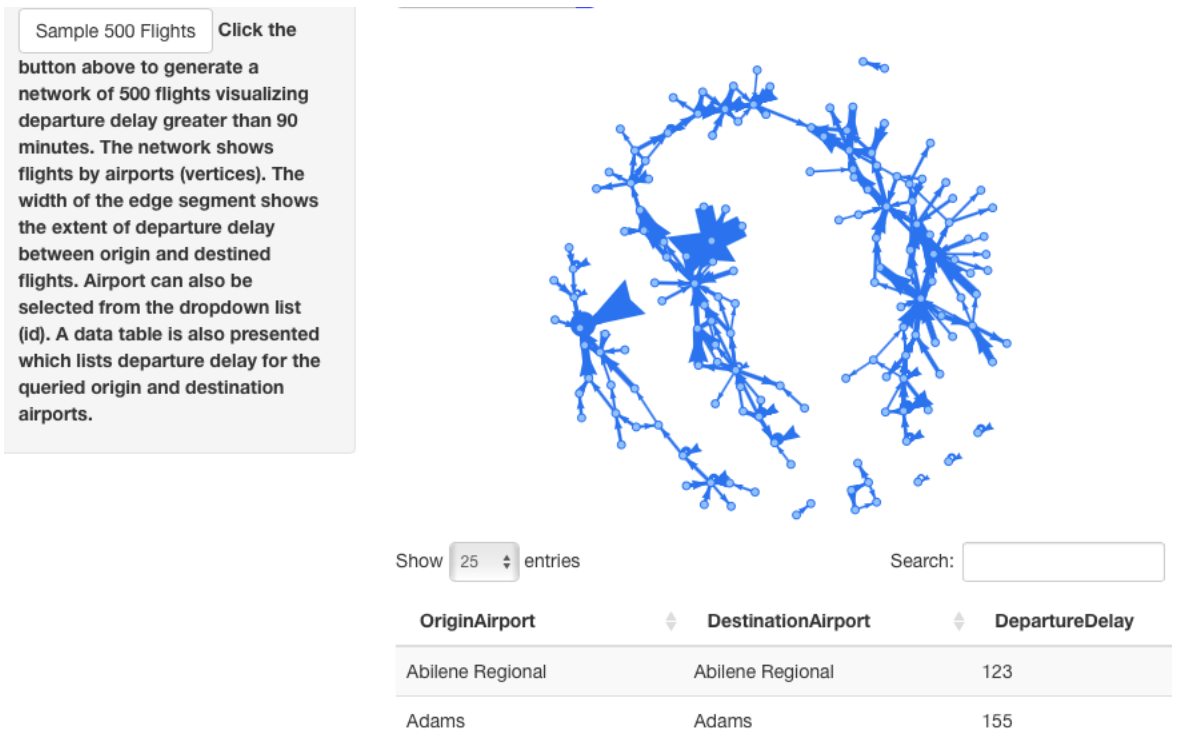
over the months were not observed though there was a presence of a general trend towards decreased total delay with an increase in pressure.

Total delay by season and weather:

In regards to seasonal variations, there appeared to be a higher increase in total delays with an increase in temperature during winter than during fall and spring. Increase in dewpoint produced a higher increase in total delays for February and March than for other months. Differences in total delay by month did not appear to be distinguishable for pressure and humidity measures. Overall trend pointed to an increase in total delay with an increase in humidity and a decrease in total delay with an increase in pressure for all months.

3.3.4 Flights Network Analysis

Fourth panel in shiny showed a network of randomly sampled 500 flights with their destination and origin airports specified as vertices and the width of the edges representing the extent of the departure delay. The network only included flights with delays greater than 90 minutes. Airports can be highilghted on the network from the dropdown menu. In addition, the data table for the network was shown which could be queried for the numerically exact departure delay. This graph was constructed with igraph and the visnetwork package. The departure delays were normalized to show appropriate scaling. An output from shiny for a random network as well as the associated data table is shown by Figure 3.4.



Click the button above to generate a network of 500 flights visualizing departure delay greater than 90 minutes. The network shows flights by airports (vertices). The width of the edge segment shows the extent of departure delay between origin and destined flights. Airport can also be selected from the dropdown list (id). A data table is also presented which lists departure delay for the queried origin and destination airports.

Show 25 entries Search:

OriginAirport	DestinationAirport	DepartureDelay
Abilene Regional	Abilene Regional	123
Adams	Adams	155

Figure 3.4: Flights Network

3.3.5 Mapping Flights

Last shiny panel showed the mean departure delay by airport from 2008 to 2016 in the United States for all flights with departure delay greater than 90 minutes. Hovering over the points displayed the state, airport as well as the average departure delay for that airport. From the graph, Wyoming Cheyenne Regional Airport appeared to have a visibly high mean departure delay of 247.71 minutes. Other airports with high departure delay included Greater Rockford Airport in Illinois with a mean departure delay of 240.22 minutes and Bemidji Regional Airport in Minnesota with mean departure delay of 235.76 minutes. Overall trend pointed to low departure delays along the West coast. An output from the shiny app is shown by Figure 3.5.

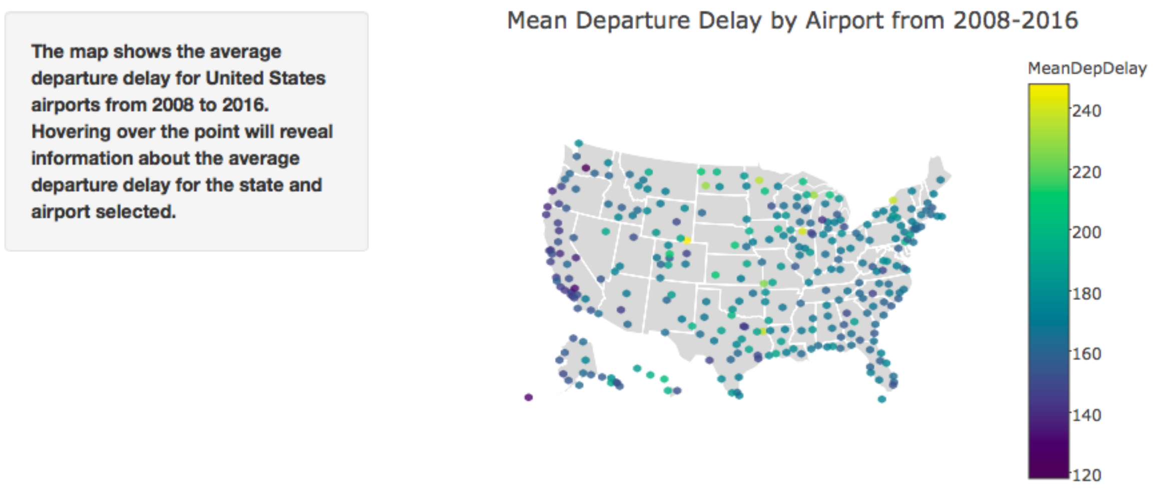

The map shows the average departure delay for United States airports from 2008 to 2016. Hovering over the point will reveal information about the average departure delay for the state and airport selected.

Figure 3.5: Maps Analysis

Chapter 4

Logistical Reasoning

4.1 Introduction

H2O platform was used to construct a logistic model to predict occurrence of departure delay over 20 minutes from 2008 to 2016. Delay of 30 minutes was chosen since it provided adequate sample for flights both experiencing and not experiencing departure delay. Analysis was performed on a random sample of 200000 observations. This size was easily transferable to the Spark environment while larger sample sizes crashed the server. Occurrence of delay over 30 minutes was assessed against year, arrival delay, carrier, air time, distance, week and season predictors. Initially, hour, month and weekend status predictors were also used to estimate the incidence of delay though they appeared to have no importance and therefore were not used in subsequent analysis.

4.2 H2O Connection

After the installation process, h2o.init() was used to establish H2O connection to local host at port 54321. Default connection is established with 1GB of memory. Additionl cluster memory can be allocated with max_mem_size specification. Since this project was performed using the Apache Spark platform, connection with YARN Hadoop cluster was established. This connection entailed a cluster initialization in the shell followed by a Spark connection as shown below.

```r
library(sparklyr)
library(rsparkling)
library(dplyr)
options(rsparkling.sparklingwater.version = "1.6.8")

#Initialize a cluster (without Hadoop connection)
h2o.init()

#Connect to YARN through shell
```

```r
kinit()
klist

#Connect to Apache Spark Hadoop in markdown
sc <- spark_connect(master = "yarn-client")
```

4.3 Spark Data Integration

Once the connection was estabished, flights dataset was created (see Appendix 2 for full details on the dataset creation). Final flights dataset creation involved copying data from Hadoop to Spark environment from 2008 to 2016. The dataset columns were then renamed. The final dataset was saved in HadoopLogMod.Rda and copied in the Spark environment with subsequent use. Data copy process from Hadoop to Spark is shown below. If a local connection was established instead of Hadoop, R data frame can be converted to H2OFrame using the as.h2o command.

```r
#If connection is established to Hadoop:
load("HadoopLogMod.Rda")
set.seed(134)
sample <- FullDatLog[sample(nrow(FullDatLog),
                              200000, replace = FALSE,
                              prob = NULL),]
LogDataMod <- copy_to(sc, sample, "LogData",
                      overwrite = TRUE)


#If no connection established to Hadoop:
#Read from local file
FlightsDat = h2o.importFile(localH2O, path = prosFlights)
#Convert to h2o data frame
FlightsDat <- as.h2o(sample)
```

4.4 Data Partitioning

After the data was copied in the Spark environment, it was partitioned in test and training sets. A 75/25 partition was used as shown below where training set had about 75% of the data (149789 observations) and test set had about 25% (50211 observations). The partition was not strictly 75/25 since the split is not performed exactly.

```r
#Partitioning data frame in Spark
partitions <- LogDataMod %>%
  sdf_partition(training = 0.75, test = 0.25, seed = 1099)

#Partitioning data locally within the H2O platform
splits <- h2o.splitFrame(LogDataMod, c(0.75,0.25), seed=1099)
```

4.5 Checking Conditions

Since the response (dep_delayIn) was a binary predictor indicating the incidence of departure delay of 30 minutes, linearity was assumed. Randomness and independence may not necessarily be valid assumptions since typically a late flight results in subsequent delays. Analysis of the randomness and independence assumption will require tracking flight schedules from 2008 to 2016. Future studies could assess this assumption. This study proceeded with caution.

4.6 Modeling

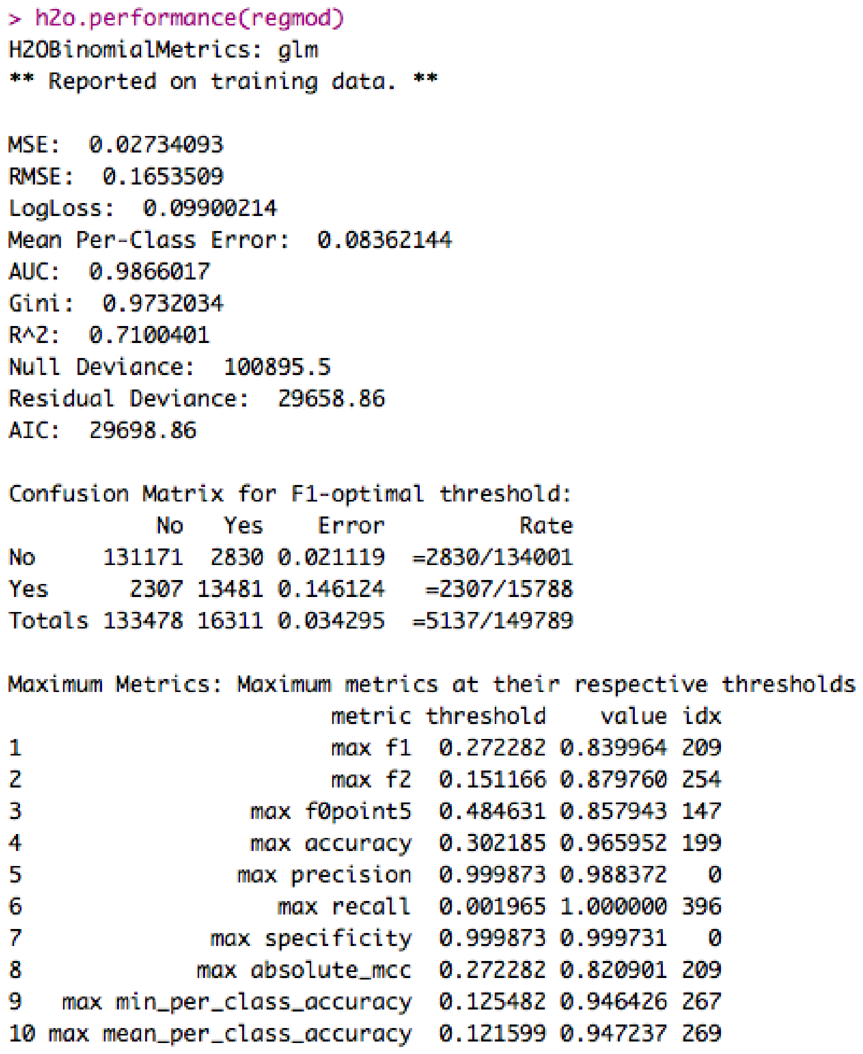
Once the data was partitioned in test and training sets, logistic regression model was specifed as shown. The setdiff command took the set difference between predictors in the training dataset and the set of predictors specified in the command (dep_DelayIn, orig_id, hour, month, weekend). The h2o.glm function can be used to specify a binomial family function. Additional arguments in the h2o.glm function include nfolds (specifies the number of folds for cross validation), alpha (0-1 numeric that specifies the elastic-net mixing parameter, set to ensure regularization and consequently prevent overfitting), lambda (specifies a non-negative shrinkage parameter) and lambda_search (logical indicating whether or not search is conducted over the lambda space specified). In addition, h2o models can be stopped early with specification of stopping metrics like misclassification error, rsquared and mean squared error. Every model has an associated model id which can be referenced for future model iterations. In the model below, a 5-fold cross validation was performed with alpha level of 0.1. As mentioned above, the logistic model below attempted to predict the occurrence of departure delay by variables including year, arrival delay, carrier, air time, distance, week and season.

```
myX = setdiff(colnames(training), c("dep_delayIn",
                                    "orig_id", "hour",
                                    "month", "weekend"))

regmod <- h2o.glm(y = "dep_delayIn", x = myX,
                  training_frame = training, family = "binomial",
        alpha = 0.1, lambda_search = FALSE, nfolds = 5)
```

4.7 Model Assessment

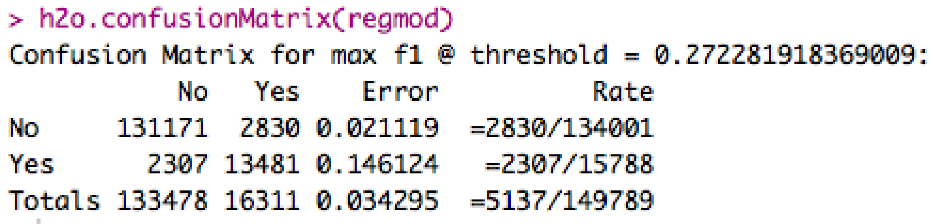
Model performance was assessed with the h2o.performance function, which provides access to evaluation metrics like MSE, RMSE, LogLoss, AUC and R^2. The R^2 for this model was 0.710 and the AUC was 0.987 as shown by Figure 4.1. The R^2 value indicated that about 71% of the variation in departure delay was accounted for by predictors year, arrival delay, carrier, air time, distance, week and season. In addition to the h2o.performance function, h2o.auc and h2o.confusionMatrix (see Figure 4.2) was used to retrieve the analogous parameters. The AUC curve was visualized as shown by figure Figure 4.3 with the plot(h2o.performance) command.

```
h2o.performance(regmod)
h2o.auc(regmod)
h2o.confusionMatrix(regmod)
accuracy <- (mat$No[1]+mat$Yes[2])/(mat$No[1]+
                                      mat$No[2]+mat$Yes[1]+
                                      mat$Yes[2])
plot(h2o.performance(regmod)) #plot the auc curve
```



```
> h2o.performance(regmod)
H2OBinomialMetrics: glm
** Reported on training data. **

MSE:  0.02734093
RMSE:  0.1653509
LogLoss:  0.09900214
Mean Per-Class Error:  0.08362144
AUC:  0.9866017
Gini:  0.9732034
R^2:  0.7100401
Null Deviance:  100895.5
Residual Deviance:  29658.86
AIC:  29698.86

Confusion Matrix for F1-optimal threshold:
           No   Yes    Error           Rate
No     131171  2830 0.021119  =2830/134001
Yes      2307 13481 0.146124   =2307/15788
Totals 133478 16311 0.034295  =5137/149789

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold    value idx
1                       max f1  0.272282 0.839964 209
2                       max f2  0.151166 0.879760 254
3                 max f0point5  0.484631 0.857943 147
4                 max accuracy  0.302185 0.965952 199
5                max precision  0.999873 0.988372   0
6                   max recall  0.001965 1.000000 396
7              max specificity  0.999873 0.999731   0
8             max absolute_mcc  0.272282 0.820901 209
9   max min_per_class_accuracy  0.125482 0.946426 267
10 max mean_per_class_accuracy  0.121599 0.947237 269
```

Figure 4.1: Model Performance



```
> h2o.confusionMatrix(regmod)
Confusion Matrix for max f1 @ threshold = 0.272281918369009:
           No   Yes    Error           Rate
No     131171  2830 0.021119  =2830/134001
Yes      2307 13481 0.146124   =2307/15788
Totals 133478 16311 0.034295  =5137/149789
```

Figure 4.2: Confusion Matrix

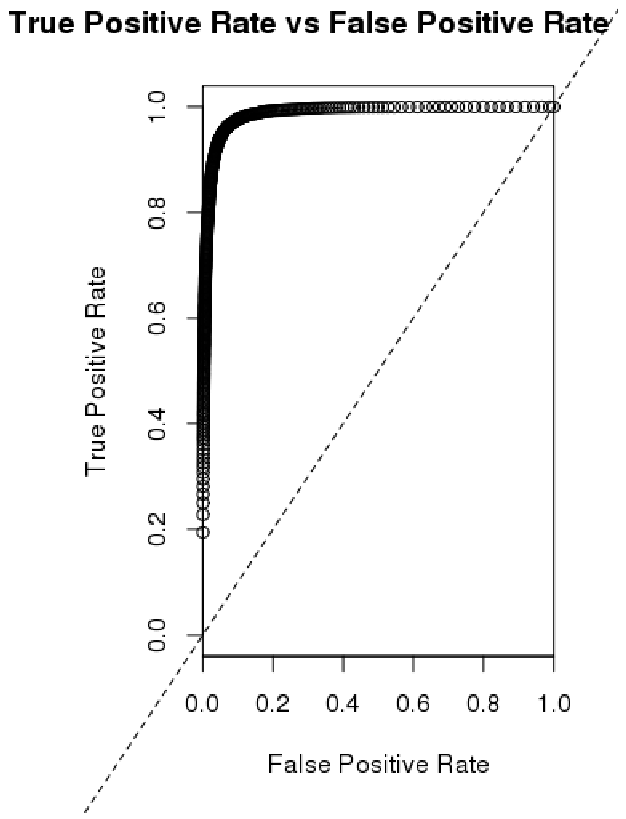


Figure 4.3: AUC Curve

As the variable importance plot in Figure 4.4 shows, arrival delay was most important in predicting the occurrence of departure delays greater than 30 minutes. Arrival delay was followed by air time and distance predictors. Air time appears to be a negative predictor of delay indicating that flights with high air time appear less likely to experience departure delays greater than 30 minutes. In comparison, flights that cover more distance are more likely to experience departure delays greater than 30 minutes. Southwest (WN) and United Airlines (UA) were more important at predicting departure delays greater than 30 minutes while US Airways (US) did not. Year 2008 and 2009 negatively predicted departure delays greater than 30 minutes while year 2015 appeared to be more likely to experience departure delays greater than 30 minutes. In comparison, summer appeared to be an important season for predicting departure delay greater than 30 minutes.

```r
h2o.varimp(regmod) #compute variable importance
h2o.varimp_plot(regmod) #plot variable importance
```

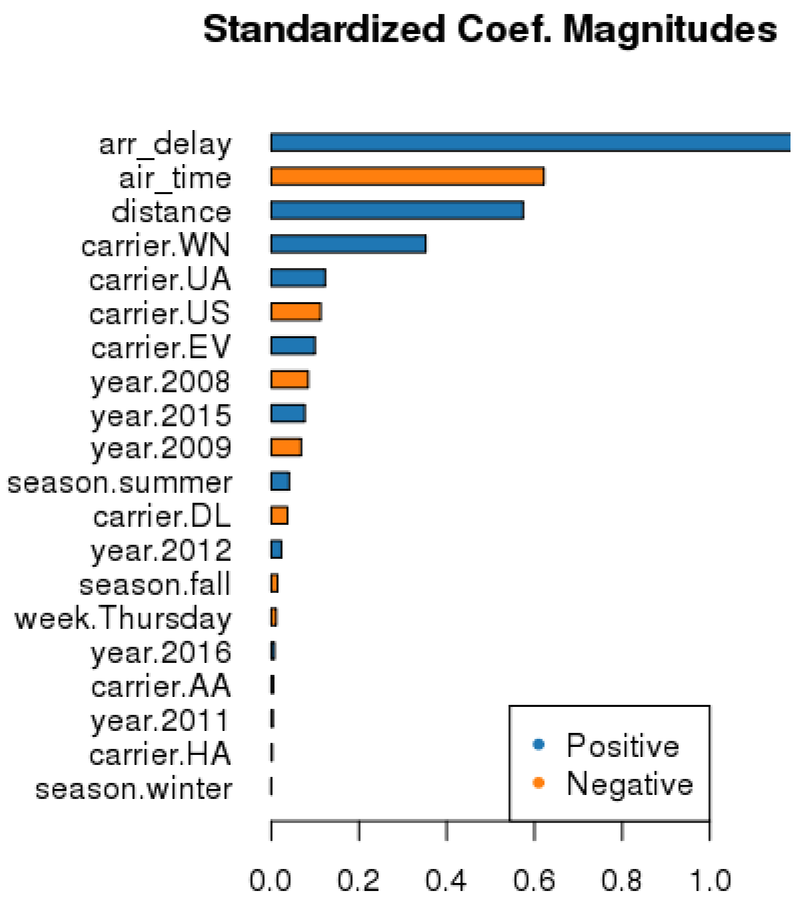


Figure 4.4: Variable Importance

4.8 Making Predictions

After model assessments were analyzed, predictions can be performed on the test set. The accuracy of the test set was calculated and compared with the accuracy of the cross-validated training set. In this case, accuracy of the test set of 0.967 compares with the accuracy of the training data of 0.966. Since the error rates are similar, occurrence of overfitting was reduced. In addition to comparing the accuracy of the test with the training set, accuracy of the training data can also be compared with the accuracy of the validation data (see Deep Learning).

```
pred <- h2o.performance(object = regmod, newdata = test)
mean(pred$predict==test$dep_delayIn) #accuracy of test set
```

4.9 Connection Shutdown

The established spark connection was disconnected with the spark_disconnect command as shown below.

```
spark_disconnect(sc) #disconnect the spark session
h2o.shutdown(prompt=FALSE) #close the H2O connection
```

4.10 Conclusion

This chapter discussed concepts like connecting to local or YARN connection, partitioning datasets and performing generalized linear regression modeling. Additionally model assessments were discussed.

Overall arrival delay appeared to be most important in predicting departure delay greater than 30 minutes indicating that flights with increased arrival delay may have increased tendency to experience departure delay. Air time and distance also appeared to be important in predicting occurrence of departure delay greater than 30 minutes. Higher air time coverage thus seemed to decrease the occurrence of departure delay suggesting that flights that have been flying for a while may have less chance of experiencing departure delay. This finding does not seem intuitive but it may be the case that if the aircraft is flying for a while, it is more adept at handling delays. More distance coverage is associated with higher likelihood of departure delay greater than 30 minutes. Summer also increased chance of departure delay which seems intuitive since it is typically a busy season for vacations as well as tourist explorations. There also appeared to be an increased likelihood of departure delay greater than 30 minutes in more recent years like 2015 than in 2008 or 2009 which could point to an increased trend in air travel resulting in increased traffic and consequently higher likelihood of experiencing departure delay.

Chapter 5

Weathering Logistics

5.1 Introduction

H2O platform was used to construct a logistic model to predict whether or not departure delay over 30 minutes can be predicted using weather data from nycflights13 package for LaGuardia, John F. Kennedy and Newark Liberty International airports in 2013. Analysis was performed on data containing 48126 observations. Occurrence of delay over 30 minutes was assessed against season, month, week, weekend, day, hour, arrival delay, distance and air time. Weather predictors included temperature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility.

5.2 Data Partitioning

After the data connection was established and data was copied in the Spark environment, it was partitioned in test and training sets. A 75/25 split of training and test data was used. The training data contained 36153 observations and test set contained 11973 observations.

```r
library(sparklyr)
library(rsparkling)
library(dplyr)
options(rsparkling.sparklingwater.version = "1.6.8")
sc <- spark_connect(master = "yarn-client")
load("flights_weather2.Rda")

partitions <- LogDataMod %>%
  sdf_partition(training = 0.75, test = 0.25, seed = 1099)

#Partitioning data locally within the H2O platform
splits <- h2o.splitFrame(LogDataMod, c(0.75,0.25), seed=1099)
```

5.3 Checking Conditions

Since the response (dep_delayIn) was a binary predictor indicating the incidence of departure delay of 30 minutes, linearity was assumed. Randomness and independence may not necessarily be valid assumptions since typically a late flight results in subsequent delays but the study proceeded with caution.

5.4 Modeling

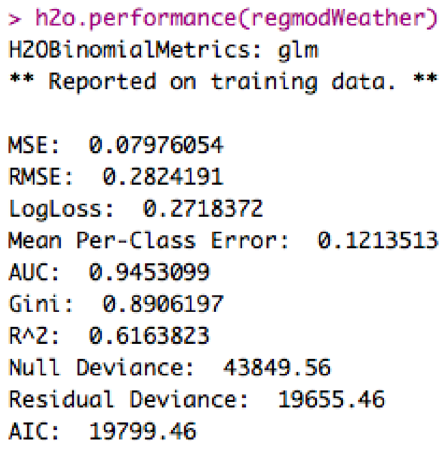
Logistic model was used as shown below to predict whether or not departure delay occurs against predictors season, month, week, weekend, day, hour, arrival delay, distance, air time, temprature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility. The same parameters as the logistic model in the previous chapter were used (alpha = 0.1, lambda_search = FALSE and 5-folds cross-validation).

```
myX = setdiff(colnames(training), c("dep_delayIn")) #set difference

regmodWeather <- h2o.glm(y = "dep_delayIn", x = myX,
                   training_frame = training, family = "binomial",
        alpha = 0.1, lambda_search = FALSE, nfolds = 5)
```

5.5 Model Assessment

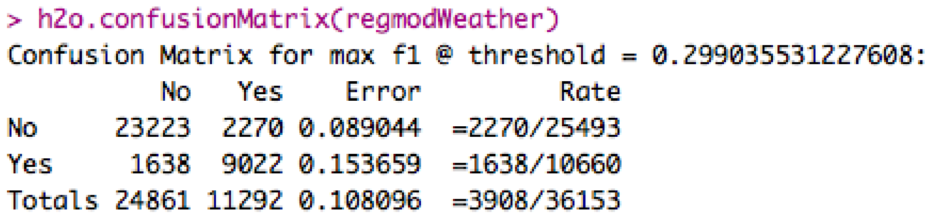
Model performance was assessed with h2o.performance. R^2 for this model was 0.616 and the AUC was 0.945 as shown by Figure 5.1. The R^2 value indicated that about 61% of the variation in the departure delay variable was accounted for by season, month, week, weekend, day, hour, arrival delay, distance, air time, temprature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility. Value of 0.61 compared with an R^2 of 0.71 for the logistic model without weather consideration. In addition to the h2o.performance function, h2o.confusionMatrix (see Figure 5.2) was used to retrieve the confusion matrix. The AUC curve was visualized as shown by figure Figure 5.3 with the plot(h2o.performance) command.

```
h2o.performance(regmodWeather)
h2o.auc(regmodWeather)
h2o.confusionMatrix(regmod)
accuracy <- (mat$No[1]+mat$Yes[2])/(mat$No[1]+
                                      mat$No[2]+mat$Yes[1]+
                                      mat$Yes[2])
plot(h2o.performance(regmodWeather)) #plot the auc curve
```



```
> h2o.performance(regmodWeather)
H2OBinomialMetrics: glm
** Reported on training data. **

MSE:  0.07976054
RMSE:  0.2824191
LogLoss:  0.2718372
Mean Per-Class Error:  0.1213513
AUC:  0.9453099
Gini:  0.8906197
R^2:  0.6163823
Null Deviance:  43849.56
Residual Deviance:  19655.46
AIC:  19799.46
```

Figure 5.1: Model Performance



```
> h2o.confusionMatrix(regmodWeather)
Confusion Matrix for max f1 @ threshold = 0.299035531227608:
          No   Yes    Error          Rate
No     23223  2270 0.089044  =2270/25493
Yes     1638  9022 0.153659  =1638/10660
Totals 24861 11292 0.108096  =3908/36153
```

Figure 5.2: Confusion Matrix

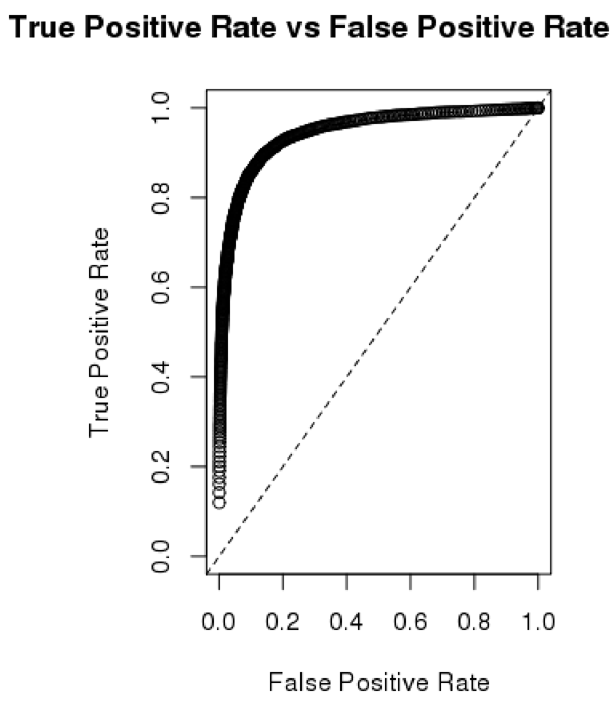


Figure 5.3: AUC Curve

As the variable importance plot in Figure 5.4 shows, arrival delay, air time and distance were all important in predicting departure delay greater than 30 minutes. While arrival delay and distance positively predicted the occurrence of departure delay, meaning an increase in these variables was linked with the occurrence of delay, distance negatively predicted delay meaning an increase in distance made the occurrence of departure delay greater than 30 minutes less likely. Day 8 of the month along with hour 6 (6 am) and 8 (8 am) were also negatively linked with the occurrence of departure delay greater than 30 minutes. Additionally American Eagle (MQ), day 14 and Tuesday were negatively associated with departure delay. Hours 19 (7 pm), 21 (9 pm) and 20 (8 pm) were positively linked with delay. Temperature and wind direction appeared to be the only two important weather predictors in the top 30 variable important plot shown. Both temperature and wind direction are positively associated meaning increase in these predictors is linked with increased likelihood of the occurrence of departure delay greater than 30 minutes.

```r
h2o.varimp(regmodWeather) #compute variable importance
h2o.varimp_plot(regmodWeather) #plot variable importance
```

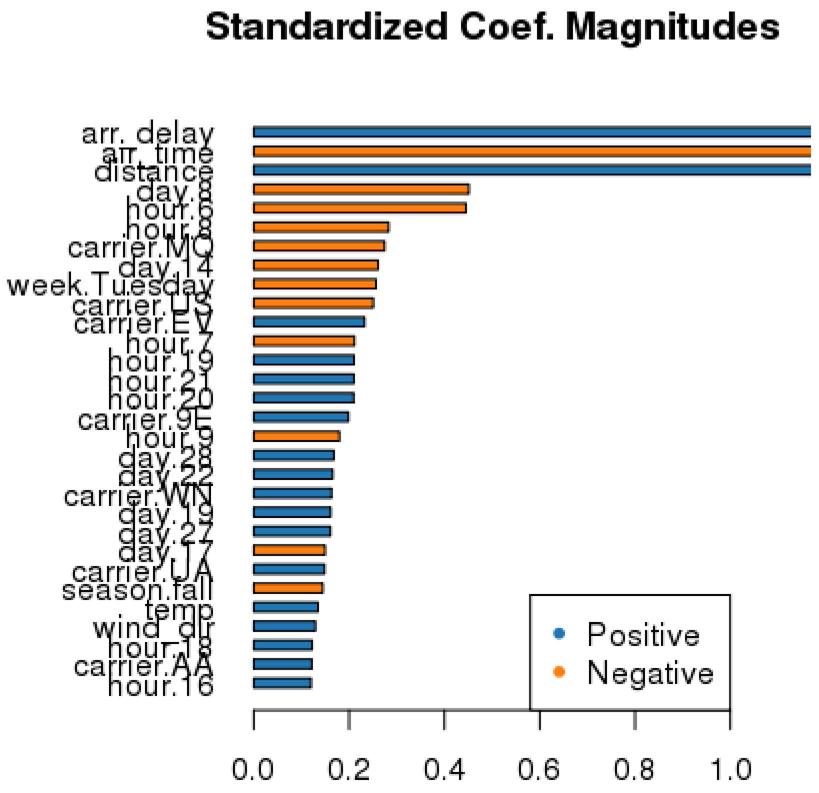


Figure 5.4: Variable Importance

5.6 Making Predictions

After model assessments were analyzed, predictions were performed on test set. The accuracy of the test set was calculated and compared with the accuracy of the cross-validated training set. In this case, accuracy of the test set of 0.891 compares with the accuracy of the training set of 0.893. Since the error rates were similar, occurrence of overfitting was reduced.

```
pred <- h2o.performance(object = regmodWeather, newdata = test)
mean(pred$predict==test$dep_delayIn) #accuracy of test set
```

5.7 Conclusion

This chapter incorporated weather gauging predictors in the logistic model built in the previous chapter. The intent was to examine the role weather metrics like temperature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility played in predicting departure delay controlling for predictors like arrival delay, distance and air time.

Overall arrival delay, distance and air time were the most important predictors. Though temperature and wind direction were the only important weather predictors, they cannot be considered extremely important since they ranked 26th and 27th, respectively. Morover the R^2 value of the logistic model with the weather variables does not appear to improve upon the logistic model built without weather predictors in the previous chapter. According to these results, weather does not appear to play a large role in predicting departure delay. Additional weather data can be collected in the future to further examine its role in predicting delays.

Chapter 6

Deep Learning

6.1 Introduction

H2O deep learning follows the multi-layer, feedforward neural network model (Reddy, n.d.). In the feedforward neural network model, the inputs are weighted, combined and transmitted as output signal by the connected neuron. Function f shown in Figure 6.1 shows a nonlinear activation function where the bias accounts for the activation threshold (Candel, Lanford, LeDell, Parmar, & Arora, 2015). A nonlinear activation function ensures that the linearly input hidden layers experience variation. Otherwise the output will simply be a linear combination of the hidden layers making the use of hidden layers irrelevant. Examples of activation functions include sigmoid and rectified linear unit (ReLU). The multi-layer platform consists of layers of interconnected neurons that is initiated with the input layer followed by layers of nonlinearity culminating in a regression or classification layer (Candel et al., 2015). An example of a multi-layer neural network is shown in Figure 6.2 (Candel et al., 2015).

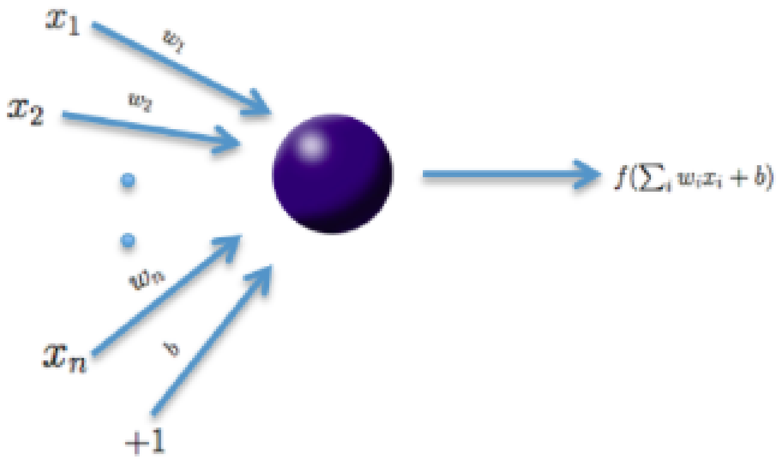


Figure 6.1: Neural Network

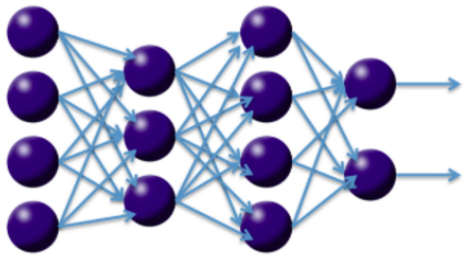


Figure 6.2: Hidden Layers

Overall H2O's deep learning functionalities include specification of regularization options, learning rate, annealing, hyperparameter optimization and model selection through grid and random search. Additionally H2O facilitates automatic processing for categorical and numerical data along with automatic imputation of missing values and ensurance of fast convergence.

H2O was used to perform deep learning to predict departure delay greater than 90 minutes. The response variable was a continuous predictor capturing extent of departure delay in minutes. This dataset differed from the dataset used to perform logistic regression since the latter contained a binary predictor to predict whether or not departure delay greater than 30 minutes occurred. In the deep learning scenario, the explanatory variables of interest included year, month, arrival delay, carrier, distance, hour, week, weekend and season.

6.2 Data Partitioning

Before data preparation and model building, connection was established to the YARN client. Following connection, data sample of 200000 was obtained given that this data size was easily transferable to the Spark environment. Since hyperparameter optimization was performed in the deep learning algorithm, a validation data set was used along with the test set for additional verification. Data split was split with 60/20/20 split consisting of 60% training, 20% validation and 20% test set.

```r
options(rsparkling.sparklingwater.version = "1.6.8")
sc <- spark_connect(master = "yarn-client")

set.seed(12)
thousand <- FullDat[sample(nrow(FullDat), 200000, replace = FALSE,
                           prob = NULL),]
mtcars_tbl <- copy_to(sc, thousand, "mtcars", overwrite = TRUE)

partitions <- mtcars_tbl %>%
  sdf_partition(training = 0.6, validation = 0.20, test = 0.20,
                seed = 1099)
```

6.3 Model Building

Following data partitioning, h2o.deeplearning function was used to predict departure delay over 90 minutes as a function of year, month, arrival delay, carrier, distance, hour, week, weekend and season. The h2o.deeplearning function includes specification of the explanatory (x) and response predictors (y). In addition, h2o.deeplearning parameters include activation function specification (Tanh, TanhWithDropout, Rectifier, RectifierWithDropout, Maxout, MaxoutWithDropout, see Figure 6.3 (Candel et al., 2015)), training and validation_frame delineation along with fine-tuning parameters like maximum model iterations, regularization parameters like l1 and l2, non-negative shrinkage parameter lambda and cross validation parameter (nfolds) specifying number of folds and model iterations. A random seed can also be specified though this is only reproducible with algorithms running on a single thread. Additionally h2o.deeplearning model provides the ability to stop the model learning early if no apparent changes in the loss function are observed.

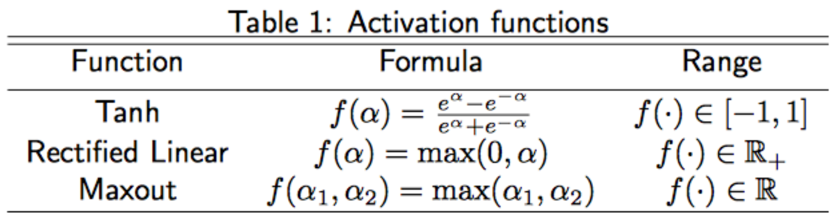


Table 1: Activation functions

Function	Formula	Range
Tanh	$f(\alpha) = \frac{e^{\alpha}-e^{-\alpha}}{e^{\alpha}+e^{-\alpha}}$	$f(\cdot) \in [-1, 1]$
Rectified Linear	$f(\alpha) = \max(0, \alpha)$	$f(\cdot) \in \mathbb{R}_+$
Maxout	$f(\alpha_1, \alpha_2) = \max(\alpha_1, \alpha_2)$	$f(\cdot) \in \mathbb{R}$

Figure 6.3: Activation Function

A simple model shown below was constructed to predict departure delay over 90 minutes as a function of the explantory variables. Epoch of one was used indicating a single data iteration. In addition, 5 folds cross validation was used. Tanh activation layer was used since it is more adept at exponentially rising functions and consequently better at containing regularization.

```r
myX = setdiff(colnames(training), ("dep_delay"))
deepmod <- h2o.deeplearning(
  y="dep_delay",
  x=myX,
  activation="Tanh",
  training_frame=training,
  validation_frame=validation,
  epochs=1,
  variable_importances=T,
  nfolds = 5,
  keep_cross_validation_predictions=T
)
```

A variable importance plot can be used to view the most important predictors produced by deepmod. In this case, a plot of the top 20 predictors was produced.

As Figure 6.4 shows, arrival delay appeared to be most important at predicting departure delay greater than 90 minutes. Following arrival delay, day status (weekday or weekend) was the next important predictor. Continental Airlines (CO), Comair, Inc. (OH) and 9 Air Co Ltd were most important carriers in predicting departure delay. Additionally hour 14 (2 pm) appeared to be important in predicting departure delay greater than 90 minutes.

```r
h2o.varimp_plot(deepmod, num_of_features = 20)
```

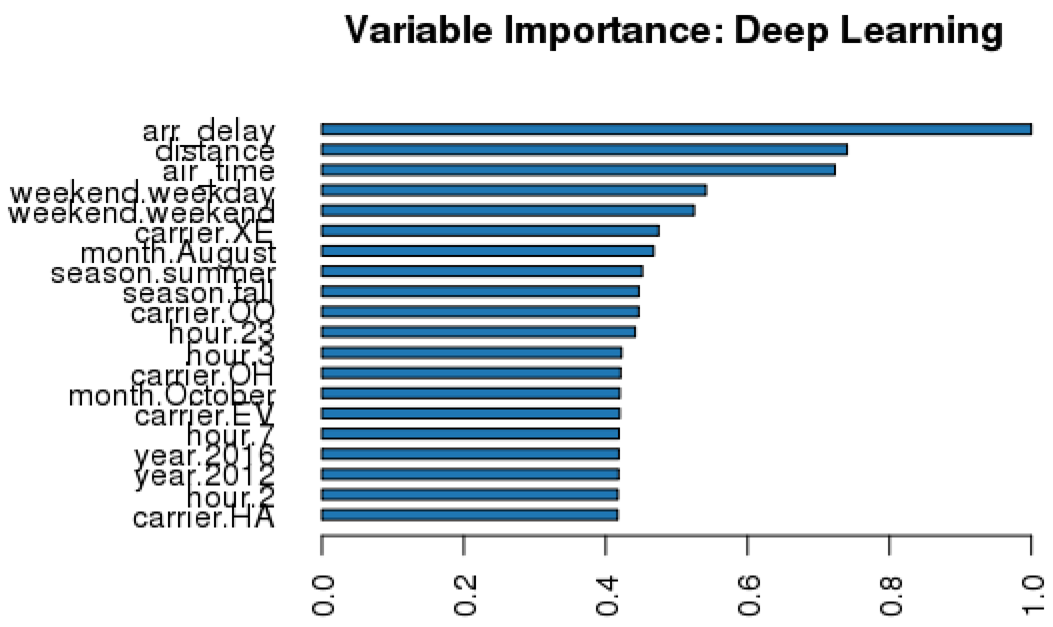


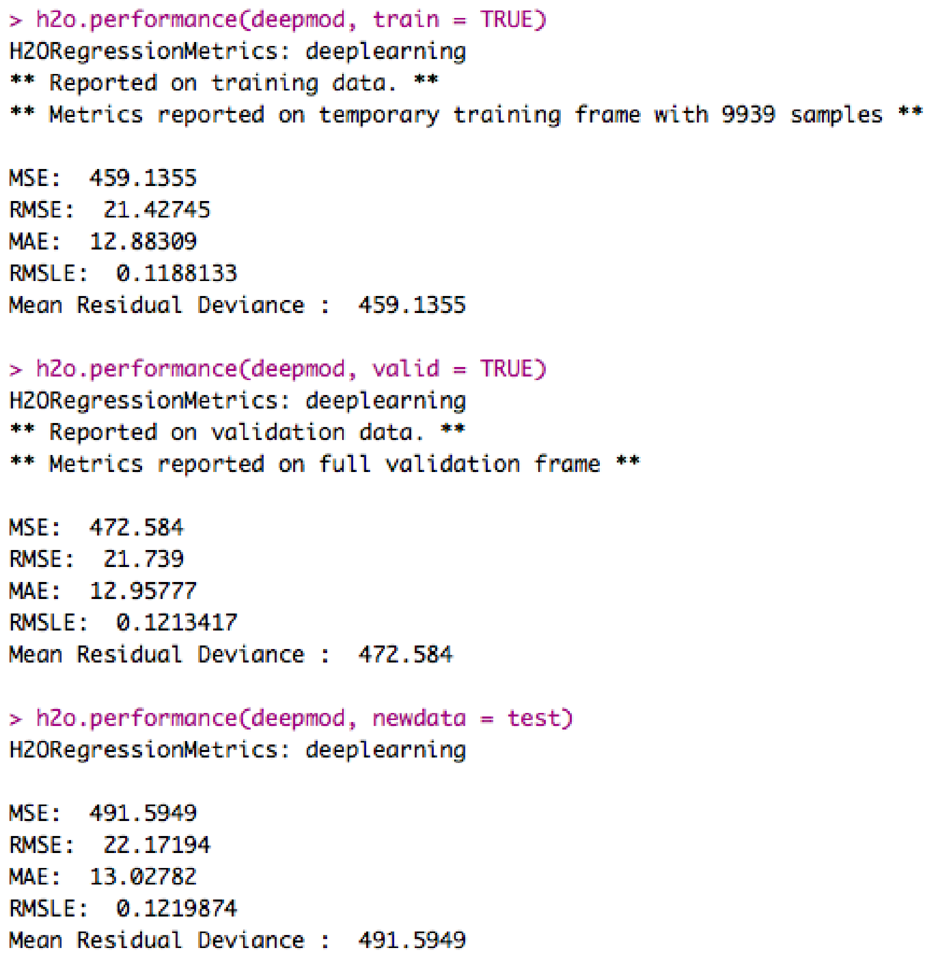
Figure 6.4: Deep Learning Variable Importance

6.4 Model Assessment

Deep learning model performance was assessed. Since the response predictor is continuous, mean squared error (MSE) was used as an error metric.

As shown by Figure 6.5, MSE on training data was 459.14 while MSE on validation data was 472.58. In comparison, MSE on test data was 491.59. Since MSE for the validation and test data is higher than MSE on training data, this was an indication of good fit. Overfitting did not appear t o be extremely visible since the difference between the MSE of the training, validation and test set did not appear to be too visible.

```
deepmod@parameters
h2o.performance(deepmod, train = TRUE)
h2o.performance(deepmod, valid = TRUE)
h2o.performance(deepmod, newdata = test)
```



```
> h2o.performance(deepmod, train = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 9939 samples **

MSE:  459.1355
RMSE:  21.42745
MAE:  12.88309
RMSLE:  0.1188133
Mean Residual Deviance :  459.1355

> h2o.performance(deepmod, valid = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on full validation frame **

MSE:  472.584
RMSE:  21.739
MAE:  12.95777
RMSLE:  0.1213417
Mean Residual Deviance :  472.584

> h2o.performance(deepmod, newdata = test)
H2ORegressionMetrics: deeplearning

MSE:  491.5949
RMSE:  22.17194
MAE:  13.02782
RMSLE:  0.1219874
Mean Residual Deviance :  491.5949
```

Figure 6.5: Deep Learning Model Performance

6.5 Saving Model

Once the model was constructed, it was saved using the h2o.saveModel command as shown below with the path specification.

```
DeepModel <- h2o.saveModel(m1, path = "/home/ajavaid17", force = FALSE)
```

Model can be loaded with the h2o.loadModel command with the specified path as an argument.

```
ld <- h2o.loadModel(path ="/home/ajavaid17/
                    DeepLearning_model_R_1487567612904_2")
```

6.6 Grid Search Model

H2O provides grid search functionality which allows the user to experiment with different hyperparameter combinations. All possible combinations of the hyperparameters are tested. In the model below, 2 different activation functions, 2 hidden layers, 2 input_dropout_ratio and 3 rate parameters were tested resulting in 24 models. Tanh and TanhWithDropout parameters were used since they better regularize for exponential functions. The hidden variable specifies the hidden layer sizes. The rate parameter specifies the learning rate where a higher rate produces less model stability and a lower rate produces slower model convergence. The rate_annealing parameter attempts to adjust learning rate.

```
hyper_params <- list(
  activation=c("Tanh", "TanhWithDropout"),
  hidden=list(c(20,20),c(40,40)),
  input_dropout_ratio=c(0,0.05),
  rate=c(0.01,0.02,0.03)
)
```

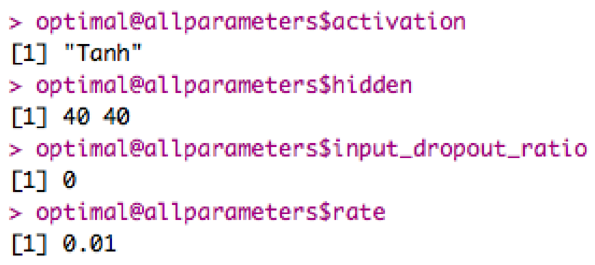
After setting up the hyperparameters, h2o.grid functionality was used to iterate over models. In order to expedite the model building process, stopping metrics were specified so the h2o.grid functionality stops when the MSE does not improve by greater than or equal to 2% (stopping_tolerance) for 2 events (stopping_rounds). In addition to the hyperparameters specified, epoch of 10 was chosen. Momentum was specified to reduce the chance of the algorithm halting at a local minima. Theoretically, momentum specification reduces terrrain irregularities thus preventing algorithm to stop at the minima (Sutskever, Martens, Dahl, & Hinton, 2013). The l1 and l2 regularization parameters attempt to prevent overfitting while the max_w2 sets the constraint for squared sum of incoming weights per unit.

```
grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id="gridDeep",
  training_frame=training,
  validation_frame=validation,
  y="dep_delay",
  x=myX,
  epochs=10,
  stopping_metric="MSE",
  stopping_tolerance=2e-2,
  stopping_rounds=2,
  score_duty_cycle=0.025,
  adaptive_rate=T,
  momentum_start=0.5,
  momentum_stable=0.9,
  momentum_ramp=1e7,
  variable_importances=T,
  l1=1e-5,
  l2=1e-5,
  max_w2=10,
  hyper_params=hyper_params
)
```

For loop can be used to iterate over all 24 models to output the MSE. Direct indexing in the grid object can be used to retrieve the optimal model along with associated parameters. The gridDeep grid search resulted in an optimal model with parameters including the Tanh activation layer, hidden layer of (40,40), input_dropout_ratio of 0 and rate of 0.01 as depicted by Figure 6.6.

```
for (model_id in grid@model_ids) {
  model <- h2o.getModel(model_id)
  mse <- h2o.mse(model, valid = TRUE)
  sprintf("Validation set MSE: %f", mse)
}

#Retrieve optimal model by MSE
grid@summary_table[1,]
optimal <- h2o.getModel(grid@model_ids[[1]])

optimal@allparameters #print all parameters of best model
h2o.performance(optimal, train = TRUE) #retrieve training MSE
h2o.performance(optimal, valid = TRUE) #retrieve validation MSE
h2o.performance(optimal, newdata = test) #retrieve test MSE
```

As shown by Figure 6.7, MSE on the optimal model for the training data was



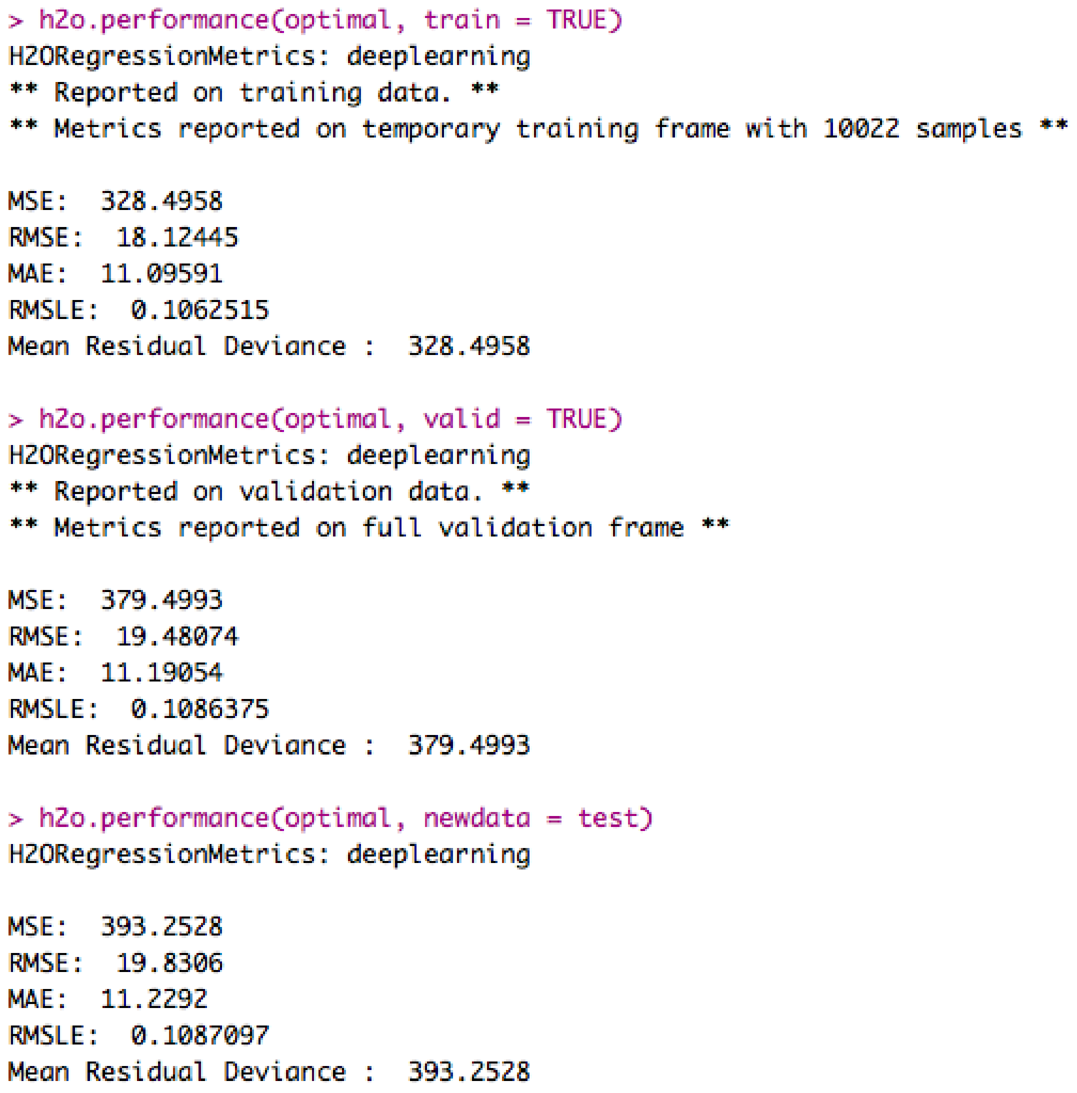
```
> optimal@allparameters$activation
[1] "Tanh"
> optimal@allparameters$hidden
[1] 40 40
> optimal@allparameters$input_dropout_ratio
[1] 0
> optimal@allparameters$rate
[1] 0.01
```

Figure 6.6: Grid Model Parameters

328.50 whereas MSE for the validation data was 379.50 and 393.25 for the test data. The validation and test errors were higher than training signaling towards a good model fit, with some reservations for overfitting. Grid search model performs better than a simple deep model since the MSE for the grid search model are lower than the MSE for deepmod (training MSE of 459.14, validation MSE of 472.58 and test MSE of 491.59).

Variable importance plot can be used to view the most important predictors. As Figure 6.8 shows, arrival delay appears to be most important in predicting departure delay greater than 90 minutes. Following arrival delay, distance and air time appear to be next important. JetSuite Air (XE) appears to be the most important carrier in predicting departure delay greater than 90 minutes. Season summer and month August additionally appear to be important in predicting departure delay greater than 90 minutes.

```
h2o.varimp_plot(optimal, num_of_features = 20)
```



```
> h2o.performance(optimal, train = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10022 samples **

MSE:  328.4958
RMSE:  18.12445
MAE:  11.09591
RMSLE:  0.1062515
Mean Residual Deviance :  328.4958

> h2o.performance(optimal, valid = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on full validation frame **

MSE:  379.4993
RMSE:  19.48074
MAE:  11.19054
RMSLE:  0.1086375
Mean Residual Deviance :  379.4993

> h2o.performance(optimal, newdata = test)
H2ORegressionMetrics: deeplearning

MSE:  393.2528
RMSE:  19.8306
MAE:  11.2292
RMSLE:  0.1087097
Mean Residual Deviance :  393.2528
```

Figure 6.7: Grid Model Performance

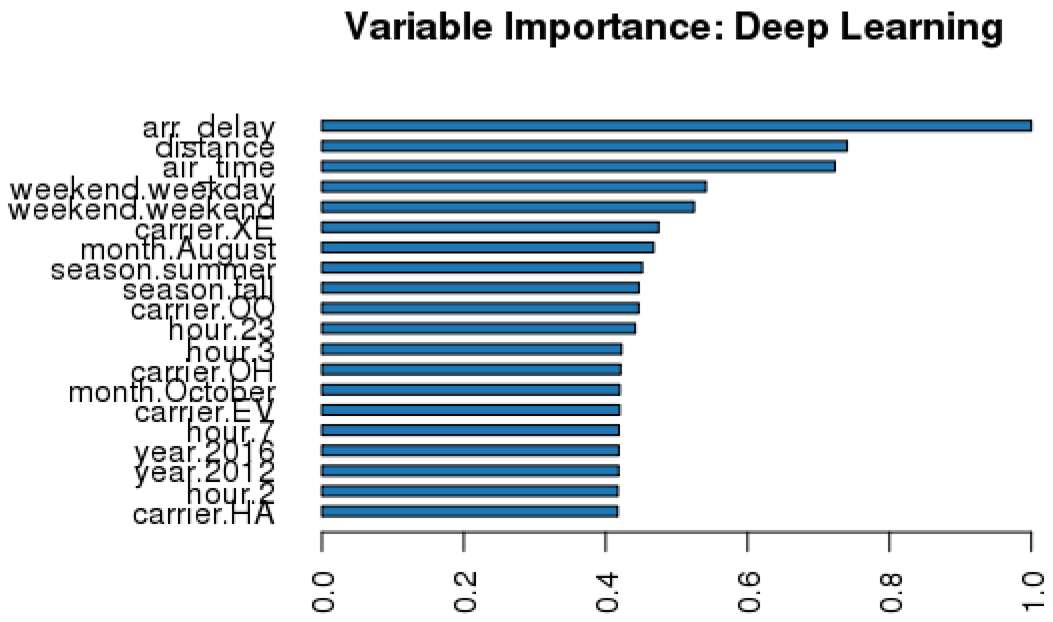


Figure 6.8: Grid Search Variable Importance

6.7 Random Grid Search Model

In comparison to grid search which iterated over combinations exhaustively and sequentially, random grid search model can be used to accelerate the process of hyperparameter selection. This is particularly useful in situations where the user wants to consider a large number of hyperparameters. Random grid search proceeds to randomly search the user specified space using an established search criteria. Since random grid search model was used, additional hyperparameters were assessed for analysis. As shown below, Tanh and TanhWithDropout functions were tested. The hidden layer additionally included (30,30,30), (50,50) and (70,70). Rate 0.03 was also tested along with 0.0 and 0.02. In addition, different combinations of regularization parameters (l1 and l2) were tested.

```
hyper_params <- list(
  activation=c("Tanh","TanhWithDropout"),
  hidden=list(c(20,20),c(30,30,30),c(40,40,40),c(50,50),c(70,70)),
  input_dropout_ratio=c(0,0.05),
  rate=c(0.01,0.02,0.03),
  l1=seq(0,1e-4,1e-6),
  l2=seq(0,1e-4,1e-6)
)
```

With the hyperparameters specified, next step included defining the search criteria. As the criteria below shows, the algorithm was defined to stop when the top 5 models were within 2% of each other. Max model running time was 600 seconds (10 minutes). In addition to the max running time, number of max_models can also be specified.

```
search_criteria = list(strategy = "RandomDiscrete",
                       max_runtime_secs = 600, max_models = 100,
                       seed=22, stopping_rounds=5,
                       stopping_tolerance=2e-2)
```

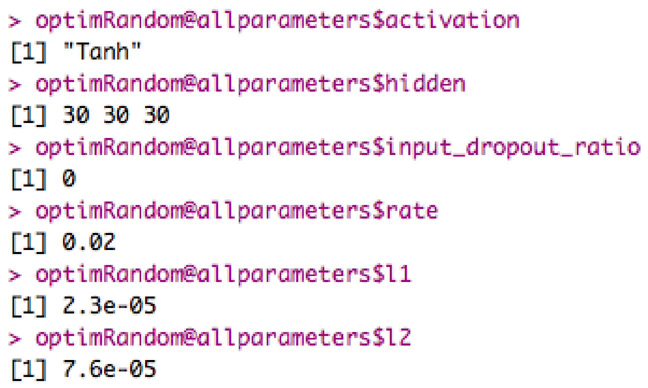
Following delineation of the search criteria, the h2o.grid function can be used to specify additional fixed parameters. These include definition of epochs of 10, max_w2 of 10, score_validation_samples of 10000 and score_duty_cycles of 0.025. The score_validation_samples specify the number of validation set samples for scoring while the score_duty_cycles specifies the maximum duty cycle fraction for scoring. The same stopping parameters as the grid search model were used. The algorithm was thus indicated to stop when the MSE does not improve by at least 2% for 2 scoring events.

```
random_grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id = "Gridrandom",
  training_frame=training,
  validation_frame=validation,
  x=myX,
  y="dep_delay",
  epochs=10,
  stopping_metric="MSE",
  stopping_tolerance=2e-2,
  stopping_rounds=2,
  score_validation_samples=10000,
  score_duty_cycle=0.025,
  max_w2=10,
  hyper_params = hyper_params,
  search_criteria = search_criteria
)
```

The validation set MSE for all the random search models can be printed with a for loop. Additionally, the best model and its associated parameters can be viewed. As shown by Figure 6.9, optimal model generated by random grid search has an activation function of Tanh, hidden layer of (30,30,30), input_dropout_ratio of 0, rate of 0.02, l1 of 2.3e-05 and l2 of 7.6e-05.

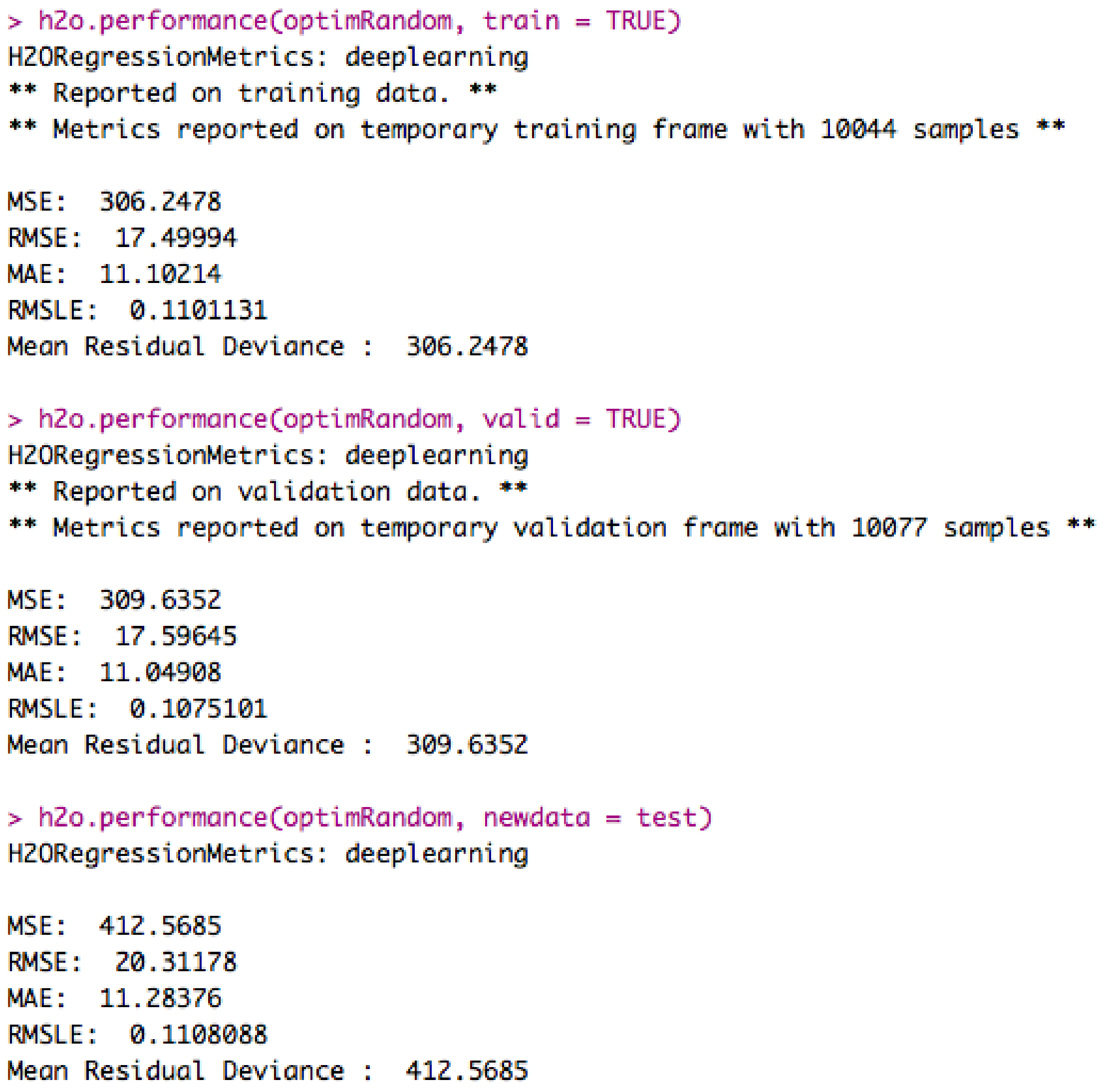
```
for (model_id in grid@model_ids) {
  model <- h2o.getModel(model_id)
  mse <- h2o.mse(model, valid = TRUE)
  sprintf("Validation set MSE: %f", mse)
}

#Retrieve optimal model by MSE
grid@summary_table[1,]
optimal <- h2o.getModel(grid@model_ids[[1]])

optimal@allparameters #print all parameters of best model
h2o.performance(optimal, train = TRUE) #retrieve training MSE
h2o.performance(optimal, valid = TRUE) #retrieve validation MSE
h2o.performance(optimal, newdata = test) #retrieve test MSE
```

As Figure 6.10 indicates, the MSE on the optimal random model for the training data was 306.25 whereas MSE for the validation set was 309.64 and 412.57 for the test set. The training and validation error are lower than those yielded by the grid search model (328.50 for training and 379.50 for validation) while the test error for the random search model is higher than the test error for the grid search model (393.25 for grid search test model). Since the random search test error is much higher than the training MSE, there may be some presence of overfitting. The errors for the random search model are lower than the initial deep learning model deepmod (training MSE of 374.39, validation MSE of 430.06 and test MSE of 423.41).



```
> optimRandom@allparameters$activation
[1] "Tanh"
> optimRandom@allparameters$hidden
[1] 30 30 30
> optimRandom@allparameters$input_dropout_ratio
[1] 0
> optimRandom@allparameters$rate
[1] 0.02
> optimRandom@allparameters$l1
[1] 2.3e-05
> optimRandom@allparameters$l2
[1] 7.6e-05
```

Figure 6.9: Random Grid Model Parameters



```
> h2o.performance(optimRandom, train = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10044 samples **

MSE:  306.2478
RMSE:  17.49994
MAE:  11.10214
RMSLE:  0.1101131
Mean Residual Deviance :  306.2478

> h2o.performance(optimRandom, valid = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on temporary validation frame with 10077 samples **

MSE:  309.6352
RMSE:  17.59645
MAE:  11.04908
RMSLE:  0.1075101
Mean Residual Deviance :  309.6352

> h2o.performance(optimRandom, newdata = test)
H2ORegressionMetrics: deeplearning

MSE:  412.5685
RMSE:  20.31178
MAE:  11.28376
RMSLE:  0.1108088
Mean Residual Deviance :  412.5685
```

Figure 6.10: Random Grid Performance

Variable importance plot can be used to view the most important predictors. In this case, a plot of the top 20 predictors was produced. As Figure 6.11 shows, arrival delay appears to be the most important predictor in predicting departure delay greater than 90 minutes. Following arrival delay, air time and distance appear to be the next important predictors. Additionally hour 4 (4 am), hour 5 (5 am), hour 0 (12 am) and weekend appear to be the next important predictors. Spirit appears to be the most important carrier in predicting departure delay greater than 90 minutes.

```r
h2o.varimp_plot(optimal, num_of_features = 20)
```

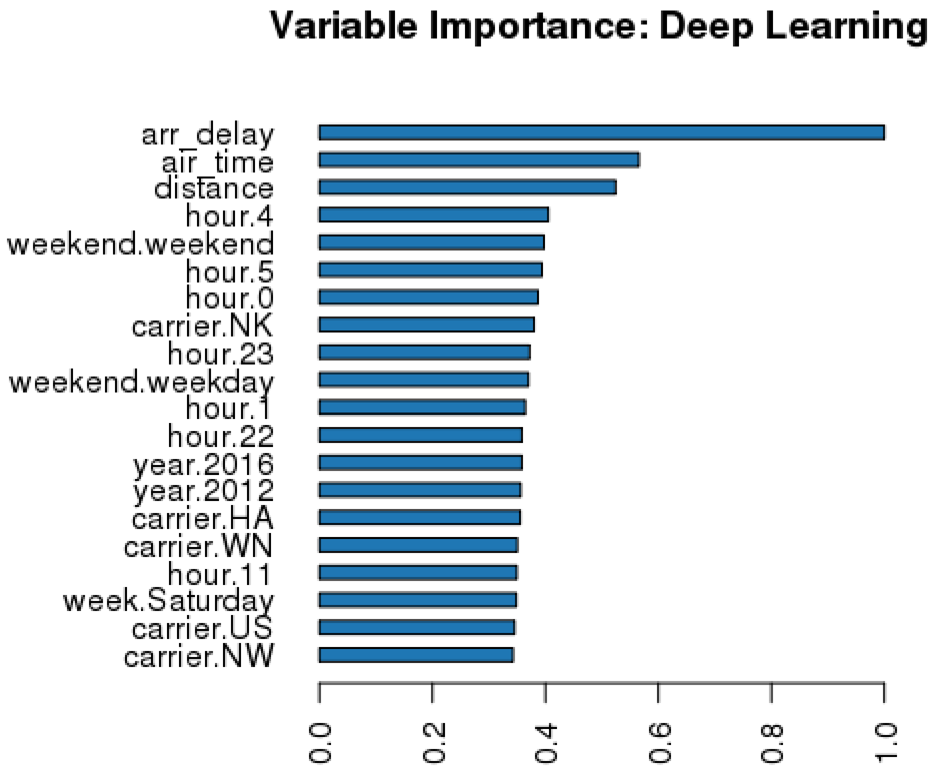


Figure 6.11: Random Grid Variable Importance

6.8 Checkpoint Model

Checkpoint functionality can be used in H2O to continue iterations from a previously built model. Checkpoint option allows specification of a previously built model key. The new model is then built as a continuation of the old model. If the model key is not supplied, then a new model is built instead. In the checkpoint model, the value of the parameters must be greater than their value set in the previous model. Parameters like activation function, max_categorical_features, momentum_ramp, momentum_stable, momentum_start and nfolds cannot be modified. A full list of all the parameters that cannot be modified can be found at (`http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/checkpoint.html`)[1].

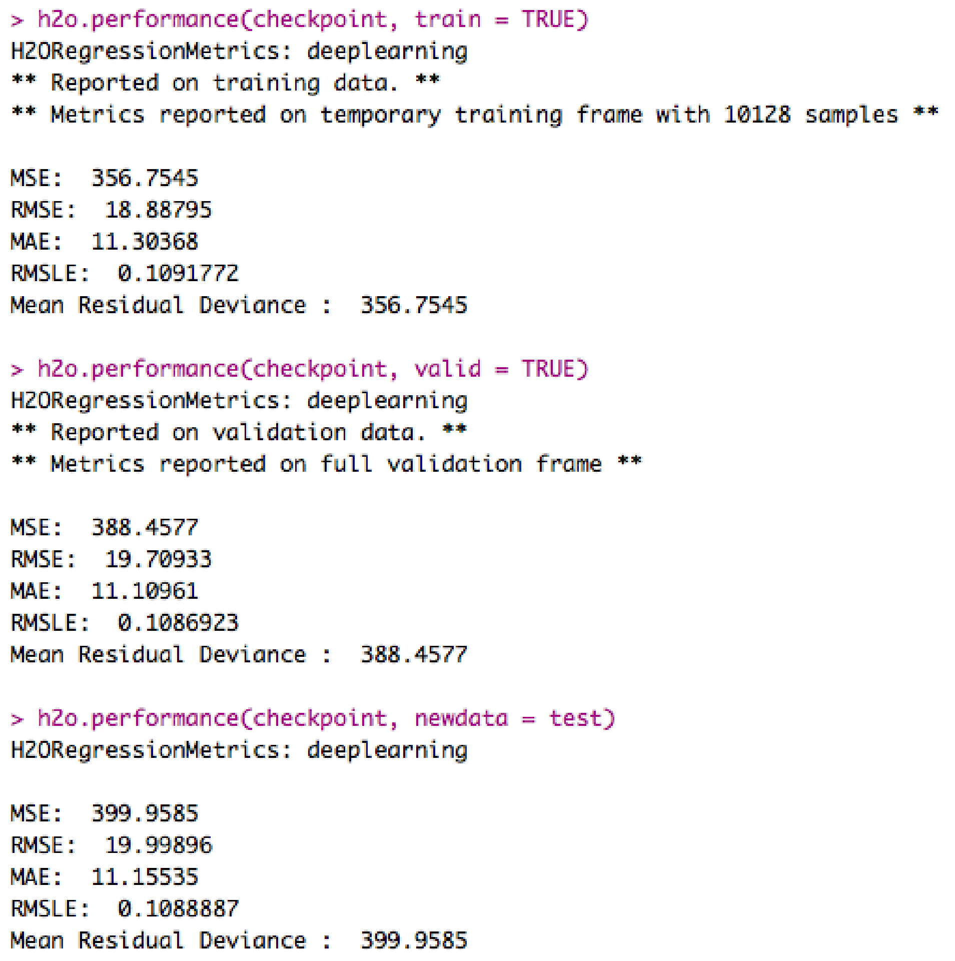
Since the test MSE (412.57) produced by the random search model was much higher than the errors on the training (306.25) and validation set (309.64), higher l1 and l2 parameters were used to produce better performance on the test set. Additionally epochs of 20 were used. These additional parameters were specified from the initial basis of the optimal random grid search model, specified below in the checkpoint specification as Gridrandom6_model_6. Same activation (Tanh), hidden layer (30,30,30) and rate (0.02) were used since these parameters can't be altered in the checkpoint model.

```r
max_epochs <- 20
checkpoint <- h2o.deeplearning(
  model_id="GridModRandom_continued2",
  activation="Tanh",
  checkpoint="Gridrandom6_model_6",
  training_frame=training,
  validation_frame=validation,
  y="dep_delay",
  x=myX,
  hidden=c(30,30,30),
  epochs=max_epochs,
  stopping_metric="MSE",
  stopping_tolerance=2e-2,
  stopping_rounds=2,
  score_duty_cycle=0.025,
  adaptive_rate=T,
  l1=1e-4,
  l2=1e-4,
  max_w2=10,
  rate = 0.02,
  variable_importances=T
)
```

As shown by Figure 6.12, MSE on the training data was 356.75, 388.46 for validation and 399.96 for the test data. The test MSE is much closer to the validation set whereas

[1](“Checkpoint,” n.d.)

MSE previously on the random grid search test data was 412.57. Since the test MSE is closer to the training and validation set's MSE, this reduces chance of overfitting.



```
> h2o.performance(checkpoint, train = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10128 samples **

MSE:  356.7545
RMSE:  18.88795
MAE:  11.30368
RMSLE:  0.1091772
Mean Residual Deviance :  356.7545

> h2o.performance(checkpoint, valid = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on full validation frame **

MSE:  388.4577
RMSE:  19.70933
MAE:  11.10961
RMSLE:  0.1086923
Mean Residual Deviance :  388.4577

> h2o.performance(checkpoint, newdata = test)
H2ORegressionMetrics: deeplearning

MSE:  399.9585
RMSE:  19.99896
MAE:  11.15535
RMSLE:  0.1088887
Mean Residual Deviance :  399.9585
```

Figure 6.12: Checkpoint Performance

According to the variable importance chart shown by Figure 6.13, arrival delay appears to be most important in predicting departure delay greater than 90 minutes. Hour 4 (4 am), air time, hour 0 (12 am) and hour 5 (5 am) appear to be next important. Additionally Spirit appears to be an important carrier in predicting departure delay greater than 90 minutes.

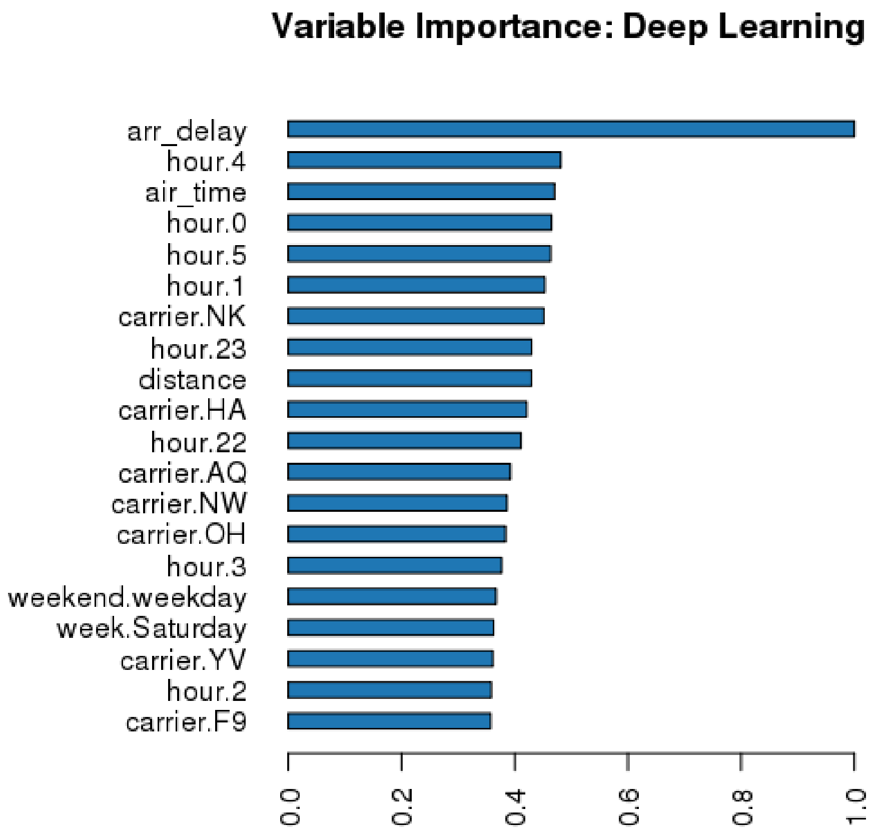


Figure 6.13: Checkpoint Variable Importance

6.9 Conclusions

This chapter discussed deep learning models including concepts like setting up hyperparameters for deep learning models through the grid search and random search methods. In addition, the checkpoint model functionality was discussed.

Overall, arrival delay appears to be most important in predicting departure delay greater than 90 minutes which indicates that airlines experiencing arrival delay appear to have a higher likelihood of experiencing departure delays. Additional important predictors include amount of distance covered as well as the amount of air time. Early morning hours between 12-5 am and weekend also appear to be important predictors of departure delay greater than 90 minutes, a trend that was also observed in the shiny app exploratory analysis.

In regards to model performance, the grid search model improved upon the original simple deep learning model. The random search model produced similar and in some regards better results than the grid search model (lower MSE on training and validation sets). Checkpoint model improved upon the random search model by reducing the test set MSE, thus lowering incidence of overfitting.

Conclusion

This project involved navigation of both the R and Hadoop servers. The h2o and rsparkling packages were explored amongst others. Shiny application was used for initial data and network exploration. H2O platform was used to perform logistic modeling and deep learning to predict the occurrence of departure delay greater than 30 minutes (binary outcome) and extent of departure delay greater than 90 minutes (continuous outcome). Additionally weather was incorporated for logistical analysis. Web scraping was also performed to extract full airport and state names. Conceptually, this project explored topics like big data infrastructure of technologies like Hadoop and Apache Spark, neural networks and feed-forward learning and machine learning hyperparameter optimization through grid and random search models.

6.10 Limitations

One limitation of this study was that only sample of 200000 was copied in the Spark environment although the original dataset had about a million observations. Capacity of Spark can be improved to handle additional data. Parallel computing through the parallel package can also be pursued to explore how the data can be split in smaller datasets with computations performed on each in a separate Spark connection and later aggregated/merged. Additionally my model shows an increase in departure delay overtime from 2008 to 2016 which seems counterintuitive. It may be worthwhile to examine this finding further and assess whether there are potential confounders clouding the result.

6.11 Future Work

Future extensions of this project can include building data pipelines to dynamically collect weather data for the associated flights and store the data in a Hadoop database. Additionally data containing security alerts can be used to gauge whether flight delays have been associated with increased alerts. Data on computer glitches and technical abnormalities as related to departure delay would also be insightful. In the future, data on plane manufacturer can be obtained to gauge whether or not the extent of departure delay experienced by a plane is associated with its manufactured date. In the future, tracing the departure delay experienced by a flight can be analyzed to observe whether or not the extent of delay is related to delay experienced by the connecting flight. Additionally other machine learning models can be explored besides feed-forward neural network employed in deep learning. These methods include SVMs, gradient boosters and ensemble modeling.

Appendix A

The First Appendix

There were no hidden code chunks in the paper. See the second appendix for accessory code.

Appendix B

The Second Appendix, for Fun

Creation of the FullDat dataset (flights data from 2008 to 2016)

```
library(sparklyr)
library(rsparkling)
library(dplyr)
library(h2o)

options(rsparkling.sparklingwater.version = "1.6.8")

sc <- spark_connect(master = "yarn-client") #connecting to the cluster.
#spark_disconnect(sc) disconnect to cluster

#loading in flights dataset from 2008 to 2016
flights2008 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights
                /flights/2008/part-m-*", header=FALSE, memory=FALSE)
flights2009 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights
                /flights/2009/part-m-*", header=FALSE, memory=FALSE)
flights2010 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights
                /flights/2010/part-m-*", header=FALSE, memory=FALSE)
flights2011 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights
                /flights/2011/part-m-*", header=FALSE, memory=FALSE)
flights2012 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights
                /flights/2012/part-m-*", header=FALSE, memory=FALSE)
flights2013 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights
                /flights/2013/part-m-*", header=FALSE, memory=FALSE)
flights2014 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights
                /flights/2014/part-m-*", header=FALSE, memory=FALSE)
flights2015 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights
                /flights/2015/part-m-*", header=FALSE, memory=FALSE)
flights2016 <- spark_read_csv(sc, "flights", "hdfs:///stats/nycflights
                /flights/2016/part-m-*", header=FALSE, memory=FALSE)
```

```
#Columns were named
flights2008 <- flights2008 %>%
  rename(year = V1) %>%
  rename(month = V2) %>%
  rename(day = V3) %>%
  rename(dep_time = V4) %>%
  rename(sched_dep_time = V5) %>%
  rename(dep_delay = V6) %>%
  rename(arr_time = V7) %>%
  rename(sched_arr_time = V8) %>%
  rename(arr_delay = V9) %>%
  rename(carrier = V10) %>%
  rename(tailnum = V11) %>%
  rename(flight = V12) %>%
  rename(origin = V13) %>%
  rename(dest = V14) %>%
  rename(air_time = V15) %>%
  rename(distance = V16) %>%
  rename(hour = V18) %>%
  rename(minute = V19) %>%
  rename(time_hour = V20)

#repeat this step for 2009-2016.
save(flights2008, file = "Flights08.Rda")

load("Flights08.Rda")
load("Flights09.Rda")
load("Flights10.Rda")
load("Flights11.Rda")
load("Flights12.Rda")
load("Flights13.Rda")
load("Flights14.Rda")
load("Flights15.Rda")
load("Flights16.Rda")

FinalDat <- rbind(DatNew08, DatNew09, DatNew10, DatNew11, DatNew12,
                  DatNew13, DatNew14, DatNew15, DatNew16)
save(FinalDat, file = "FinalDataYear.Rda")
```

H2O Logistic Regression

```r
library(sparklyr)
library(rsparkling)
library(dplyr)
library(h2o)

options(rsparkling.sparklingwater.version = "1.6.8")
sc <- spark_connect(master = "yarn-client")

log <- load("FullLogData.Rda")
datlog <- FullDatLog

DatNew <- FullDatLog
DatNew$hour <- hour(as.POSIXct(DatNew$time_hour))
DatNew$week <- weekdays(as.Date(DatNew$time_hour))
DatNew$hour <- as.numeric(DatNew$hour)
DatNew$hour <- as.factor(DatNew$hour)
DatNew$weekend<- ifelse(DatNew$week %in% c("Saturday", "Sunday"),
                        "weekend","weekday")
data3 <- DatNew
data3$carrier <- as.character(data3$carrier)


data3$weekend <- as.factor(data3$weekend)
data3$week <- as.factor(data3$week)
data3$carrier <- as.factor(data3$carrier)
data3$month <- as.factor(data3$month)
data3$month <- plyr::mapvalues(data3$month, from = c("1", "2", "3",
                                                     "4", "5", "6",
                                                     "7", "8", "9",
                                                     "10", "11", "12"),
                               to = c("January", "February", "March",
                                      "April", "May", "June", "July",
                                      "August", "September", "October",
                                      "November", "December"))
data3$season <- ifelse(data3$month %in% c("March", "April", "May"),
                       "spring",
                       ifelse(data3$month %in% c("June", "July",
                                                 "August"),
                              "summer",
                              ifelse(data3$month %in% c("September",
                                                        "October",
                                                        "November"),
                                     "fall", "winter")))
```

```r
data3$season <- as.factor(data3$season)
data3$year <- as.factor(data3$year)
FullDatLog <- data3

FullDatLog$year <- as.factor(FullDatLog$year)
FullDatLog$dep_delayIn = ifelse(FullDatLog$dep_delay > 30, "Yes","No")
FullDatLog$dep_delayIn <- as.factor(FullDatLog$dep_delayIn)
FullDatLog1 <- FullDatLog[c(1,2,9,10,15,16,18,21,22,23,24)]

FullDatLog <- FullDatLog1
save(FullDatLog, file = "HadoopLogMod.Rda")

FullDatLog2 <- load("HadoopLogMod.Rda")
set.seed(134)
sampled <- FullDatLog[sample(nrow(FullDatLog), 200000, replace = FALSE,
                             prob = NULL),]

mtcars_tbl <- copy_to(sc, sampled, "LogData", overwrite = TRUE)
partitions <- mtcars_tbl %>%
  sdf_partition(training = 0.75, test = 0.25, seed = 1099)

training <- as_h2o_frame(sc, partitions$training)
test <- as_h2o_frame(sc, partitions$test)

training$dep_delayIn <- as.factor(training$dep_delayIn)
training$season <- as.factor(training$season)
training$week <- as.factor(training$week)
training$weekend <- as.factor(training$weekend)
training$carrier <- as.factor(training$carrier)
training$hour <- as.factor(training$hour)
training$month <- as.factor(training$month)
training$year <- as.factor(training$year)


test$dep_delayIn <- as.factor(test$dep_delayIn)
test$season <- as.factor(test$season)
test$week <- as.factor(test$week)
test$weekend <- as.factor(test$weekend)
test$carrier <- as.factor(test$carrier)
test$hour <- as.factor(test$hour)
test$month <- as.factor(test$month)
test$year <- as.factor(test$year)

myX = setdiff(colnames(training), c("dep_delayIn", "orig_id", "hour",
                                    "month", "weekend"))

regmod <- h2o.glm(y = "dep_delayIn", x = myX,
                  training_frame = training, family = "binomial",
        alpha = 0.1, lambda_search = FALSE, nfolds = 5)

h2o.performance(regmod)
h2o.varimp(regmod)
h2o.varimp_plot(regmod, num_of_features = 20)
mat <- h2o.confusionMatrix(regmod)
#model accuracy
(mat$No[1]+mat$Yes[2])/(mat$No[1]+mat$No[2]+mat$Yes[1]+mat$Yes[2])

pred <- h2o.predict(object = regmod, newdata = test)
mean(pred$predict==test$dep_delayIn)
plot(h2o.performance(regmod))

#Weather Logistic Regression

flights$hour <- ifelse(flights$hour == 24, 0, flights$hour)
flights_weather <- left_join(flights, weather)
flights_weather$total <- flights_weather$dep_delay +
  flights_weather$arr_delay
flights_weather2 <- filter(flights_weather, total > 0)

DatNew <- flights_weather2
DatNew$hour <- hour(as.POSIXct(DatNew$time_hour))
DatNew$week <- weekdays(as.Date(DatNew$time_hour))
DatNew$hour <- as.numeric(DatNew$hour)
DatNew$hour <- as.factor(DatNew$hour)
DatNew$weekend<- ifelse(DatNew$week %in% c("Saturday", "Sunday"),
                        "weekend","weekday")
DatNew$month <- as.factor(DatNew$month)
DatNew$month <- plyr::mapvalues(DatNew$month, from = c("1", "2", "3",
                                                       "4", "5","6",
                                                       "7", "8", "9",
                                                       "10","11", "12"),
                                to = c("January", "February", "March",
                                       "April", "May",
                                       "June", "July", "August",
                                       "September",
                                       "October", "November",
                                       "December"))
DatNew$season <- ifelse(DatNew$month %in% c("March", "April", "May"),
                        "spring",
                        ifelse(DatNew$month %in% c("June", "July",
                                                   "August"),
                               "summer",
                               ifelse(DatNew$month %in% c("September",
                            "October", "November"), "fall", "winter")))

flights_weather2 <- DatNew
save(flights_weather2, file = "flights_weather22.Rda")

load("flights_weather22.Rda")
head(flights_weather2)
names(flights_weather2)
nrow(flights_weather2)
flights2 <- na.omit(flights_weather2)

mtcars_tbl <- copy_to(sc, flights2, "flights", overwrite = TRUE)
partitions <- mtcars_tbl %>%
  sdf_partition(training = 0.75, test = 0.25, seed = 1099)

training <- as_h2o_frame(sc, partitions$training)
test <- as_h2o_frame(sc, partitions$test)

training$dep_delayIn <- ifelse(training$dep_delay > 30, "Yes", "No")
test$dep_delayIn <- ifelse(test$dep_delay > 30, "Yes", "No")

training$dep_delayIn <- as.factor(training$dep_delayIn)
training$season <- as.factor(training$season)
training$week <- as.factor(training$week)
training$weekend <- as.factor(training$weekend)
training$carrier <- as.factor(training$carrier)
training$hour <- as.factor(training$hour)
training$month <- as.factor(training$month)
training$year <- as.factor(training$year)
training$day <- as.factor(training$day)


test$dep_delayIn <- as.factor(test$dep_delayIn)
test$season <- as.factor(test$season)
test$week <- as.factor(test$week)
test$weekend <- as.factor(test$weekend)
test$carrier <- as.factor(test$carrier)
test$hour <- as.factor(test$hour)
test$month <- as.factor(test$month)
```

```r
test$year <- as.factor(test$year)
test$day <- as.factor(test$day)

testDat <- test[,c(1,2,3,9,10,15,16,17,20,21,22,
                   23,24,25,26,27,28,30,31,32,33)]
trainDat <- training[,c(1,2,3,9,10,15,16,17,20,21,22,
                        23,24,25,26,27,28,30,31,32,33)]

myX = setdiff(colnames(testDat), c("dep_delayIn"))
regmod <- h2o.glm(y = "dep_delayIn", x = myX,
                  training_frame = trainDat,
                  family = "binomial",
                  alpha = 0.1, lambda_search = FALSE, nfolds = 5)

regmodWeather <- regmod
h2o.performance(regmodWeather)
h2o.varimp(regmod)
h2o.varimp_plot(regmodWeather, num_of_features = 30)
h2o.confusionMatrix(regmodWeather)
(mat$No[1]+mat$Yes[2])/(mat$No[1]+mat$No[2]+mat$Yes[1]+mat$Yes[2])

pred <- h2o.predict(object = regmodWeather, newdata = testDat)
mean(pred$predict==testDat$dep_delayIn) #accuracy of test set
```

H2O Deep Learning

```r
library(sparklyr)
library(rsparkling)
library(dplyr)
library(h2o)


options(rsparkling.sparklingwater.version = "1.6.8")
#spark_disconnect(sc)
sc <- spark_connect(master = "yarn-client")

yas <- load("FinalDataYear.Rda")
head(FinalDat)
nrow(FinalDat)
unique(FinalDat$year)

DatNew <- FinalDat
DatNew$hour <- hour(as.POSIXct(DatNew$time_hour))
```

```r
DatNew$week <- weekdays(as.Date(DatNew$time_hour))
DatNew$hour <- as.numeric(DatNew$hour)
DatNew$hour <- as.factor(DatNew$hour)
DatNew$weekend<- ifelse(DatNew$week %in% c("Saturday", "Sunday"),
                        "weekend","weekday")
data3 <- DatNew
data3$carrier <- as.character(data3$carrier)


data3$weekend <- as.factor(data3$weekend)
data3$week <- as.factor(data3$week)
data3$carrier <- as.factor(data3$carrier)
data3$month <- as.factor(data3$month)
data3$month <- plyr::mapvalues(data3$month, from = c("1", "2", "3",
                                                     "4", "5", "6",
                                                     "7", "8", "9",
                                                     "10", "11", "12"),
                               to = c("January", "February", "March",
                                      "April", "May", "June", "July",
                                      "August", "September",
                                      "October", "November",
                                      "December"))
data3$season <- ifelse(data3$month %in% c("March", "April", "May"),
                       "spring",
                       ifelse(data3$month %in% c("June", "July",
                                                 "August"),
                              "summer",
                              ifelse(data3$month %in% c("September",
                                                        "October",
                                                        "November"),
                                     "fall", "winter")))

data3$season <- as.factor(data3$season)
data3$year <- as.factor(data3$year)
FullDatLog <- data3
FullDatLog$year <- as.factor(FullDatLog$year)
FullDatLog$dep_delay <- as.numeric(FullDatLog$dep_delay)
nrow(FullDatLog)

set.seed(12)
sampled <- FullDatLog[sample(nrow(FullDatLog),
                             200000, replace = FALSE, prob = NULL),]
mtcars_tbl <- copy_to(sc, sampled, "deep", overwrite = TRUE)
partitions <- mtcars_tbl %>%
  sdf_partition(training = 0.5, validation = 0.25,
                test = 0.25, seed = 1099)

training <- as_h2o_frame(sc, partitions$training)
validation <- as_h2o_frame(sc, partitions$validation)
test <- as_h2o_frame(sc, partitions$test)

training$dep_delay <- as.numeric(training$dep_delay)
training$arr_delay <- as.numeric(training$arr_delay)
training$air_time <- as.numeric(training$air_time)
training$season <- as.factor(training$season)
training$week <- as.factor(training$week)
training$weekend <- as.factor(training$weekend)
training$carrier <- as.factor(training$carrier)
training$hour <- as.factor(training$hour)
training$month <- as.factor(training$month)
training$year <- as.factor(training$year)

validation$dep_delay <- as.numeric(validation$dep_delay)
validation$arr_delay <- as.numeric(validation$arr_delay)
validation$air_time <- as.numeric(validation$air_time)
validation$season <- as.factor(validation$season)
validation$week <- as.factor(validation$week)
validation$weekend <- as.factor(validation$weekend)
validation$carrier <- as.factor(validation$carrier)
validation$hour <- as.factor(validation$hour)
validation$month <- as.factor(validation$month)
validation$year <- as.factor(validation$year)

test$dep_delay <- as.numeric(test$dep_delay)
test$arr_delay <- as.numeric(test$arr_delay)
test$air_time <- as.numeric(test$air_time)
test$season <- as.factor(test$season)
test$week <- as.factor(test$week)
test$weekend <- as.factor(test$weekend)
test$carrier <- as.factor(test$carrier)
test$hour <- as.factor(test$hour)
test$month <- as.factor(test$month)
test$year <- as.factor(test$year)

training <- training[,c(1,2,6,9,10,15,16,18,21,22,23)]
test <- test[,c(1,2,6,9,10,15,16,18,21,22,23)]
validation <- validation[,c(1,2,6,9,10,15,16,18,21,22,23)]
```

```r
myX = setdiff(colnames(training), ("dep_delay"))

#First simplified deep learning model
m1 <- h2o.deeplearning(
  y="dep_delay",
  x=myX,
  activation="Tanh",
  training_frame=training,
  validation_frame=validation,
  epochs=1,
  variable_importances=T,
  nfolds = 5,
  keep_cross_validation_predictions=T
)

deepmod <- m1
h2o.performance(deepmod, train = TRUE)
h2o.performance(deepmod, valid = TRUE)
h2o.performance(deepmod, newdata = test)
h2o.varimp_plot(deepmod, num_of_features = 20)

#Grid search model iteration
hyper_params <- list(
  activation=c("Tanh", "TanhWithDropout"),
  hidden=list(c(20,20),c(40,40)),
  input_dropout_ratio=c(0,0.05),
  rate=c(0.01,0.02,0.03)
)

grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id="gridDeep",
  training_frame=training,
  validation_frame=validation,
  y="dep_delay",
  x=myX,
  epochs=10,
  stopping_metric="MSE",
  stopping_tolerance=2e-2,
  stopping_rounds=2,
  score_duty_cycle=0.025,
  adaptive_rate=T,
  momentum_start=0.5,
  momentum_stable=0.9,
  momentum_ramp=1e7,
  variable_importances=T,
  l1=1e-5,
  l2=1e-5,
  max_w2=10,
  hyper_params=hyper_params
)

for (model_id in grid@model_ids) { #print model MSE
  model <- h2o.getModel(model_id)
  mse <- h2o.mse(model, valid = TRUE)
  print(sprintf("Validation set MSE: %f", mse))
}
grid@summary_table[1,]
optimal <- h2o.getModel(grid@model_ids[[1]])

optimal@allparameters #print all parameters of best model
h2o.performance(optimal, train = TRUE) #retrieve training MSE
h2o.performance(optimal, valid = TRUE) #retrieve validation MSE
h2o.performance(optimal, newdata = test) #retrieve test MSE

h2o.varimp_plot(optimal, num_of_features = 20)

#Random grid search model
hyper_params <- list(
  activation=c("Tanh","TanhWithDropout"),
  hidden=list(c(20,20),c(30,30,30),c(40,40,40),
              c(50,50),c(70,70)),
  input_dropout_ratio=c(0,0.05),
  rate=c(0.01,0.02,0.03),
  l1=seq(0,1e-4,1e-6),
  l2=seq(0,1e-4,1e-6)
)

search_criteria = list(strategy = "RandomDiscrete",
                       max_runtime_secs = 600,
                       max_models = 100,
                       seed=22, stopping_rounds=5,
                       stopping_tolerance=2e-2)

random_grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id = "Gridrandom6_model_6",
  training_frame=training,
  validation_frame=validation,
  x=myX,
  y="dep_delay",
  epochs=10,
  stopping_metric="MSE",
  stopping_tolerance=2e-2,
  stopping_rounds=2,
  score_validation_samples=10000,
  score_duty_cycle=0.025,
  max_w2=10,
  hyper_params = hyper_params,
  search_criteria = search_criteria
)

for (model_id in grid@model_ids) {
  model <- h2o.getModel(model_id)
  mse <- h2o.mse(model, valid = TRUE)
  sprintf("Validation set MSE: %f", mse)
}

#Retrieve optimal model by MSE
grid@summary_table[1,]
optimal <- h2o.getModel(grid@model_ids[[1]])

optimal@allparameters #print all parameters of best model
h2o.performance(optimal, train = TRUE) #retrieve training MSE
h2o.performance(optimal, valid = TRUE) #retrieve validation MSE
h2o.performance(optimal, newdata = test) #retrieve test MSE

h2o.varimp_plot(optimal, num_of_features = 20)

DeepMod <- h2o.saveModel(optimal, path = "/home/ajavaid17",
                         force = FALSE)
randomModel <- h2o.loadModel(path = "/home/ajavaid17/DeepLearning_
                             model_R_1487567612904_2")

#Checkpoint continuation from random model
max_epochs <- 20
checkpoint <- h2o.deeplearning(
  model_id="GridModRandom_continued2",
  activation="Tanh",
  checkpoint="Gridrandom6_model_6",
  training_frame=training,
  validation_frame=validation,
  y="dep_delay",
  x=myX,
  hidden=c(30,30,30),
  epochs=max_epochs,
  stopping_metric="MSE",
  stopping_tolerance=2e-2,
  stopping_rounds=2,
  score_duty_cycle=0.025,
  adaptive_rate=T,
  l1=1e-4,
  l2=1e-4,
  max_w2=10,
  rate = 0.02,
  variable_importances=T
)

spark_disconnect(sc)
h2o.shutdown(prompt=FALSE)
```

Shiny Application Code

```
library(shiny)
library(dplyr)
library(mosaic)
library(base)
library(plotly)
library(ggplot2)
library(nycflights13)
library(lubridate)
library(igraph)
require(visNetwork)

options(shiny.sanitize.errors = TRUE)

data(flights)
data(weather)

dat <- load("airportFullDat.Rda")
datcar <- load("carrierData.Rda")
dat2 <- load("FinalDataYear.Rda")
map <- load("USMap.Rda")

nrow(FinalDat)
```

```
airportsVis <- read.csv("http://datasets.flowingdata.com/tuts/
                        maparcs/airports.csv", header = TRUE)
data2 <- stateFin2
DatNew <- FinalDat


DatNew$time_hour <- as.POSIXct(DatNew$time_hour)
DatNew$hour <- hour(DatNew$time_hour)
#DatNew$hour <- hour(as.POSIXct(DatNew$time_hour))
DatNew$week <- weekdays(as.Date(DatNew$time_hour))
DatNew$hour <- as.numeric(DatNew$hour)
DatNew$hour <- as.factor(DatNew$hour)
DatNew$weekend<- ifelse(DatNew$week %in% c("Saturday", "Sunday"),
                        "weekend","weekday")
data3 <- DatNew
data3$carrier <- as.character(data3$carrier)
carrierDat$codeCar <- as.character(carrierDat$codeCar)

data3 <- carrierDat %>% inner_join(data3,
                                   by = c("codeCar" = "carrier"))
data3 <- plyr::rename(data3, c("code" = "carrier"))

data3$weekend <- as.factor(data3$weekend)
data2$OriginAirport <- as.factor(data2$OriginAirport)
data2$DestAirport <- as.factor(data2$DestAirport)
data2$OriginState <- as.factor(data2$OriginState)
data2$DestState <- as.factor(data2$DestState)
data3$week <- as.factor(data3$week)
data3$carrier <- as.factor(data3$carrier)
data3$month <- as.factor(data3$month)
data3$month <- plyr::mapvalues(data3$month,
from = c("1", "2", "3", "4", "5", "6", "7", "8",
         "9", "10", "11", "12"),
                               to = c("January",
                                      "February",
                                      "March",
                                      "April",
                                      "May",
                                      "June",
                                      "July",
                                      "August",
                                      "September",
                                      "October",
                                      "November",
                                      "December"))
data3$season <- ifelse(data3$month %in% c("March",
                                          "April",
                                          "May"), "spring",
                       ifelse(data3$month %in% c("June",
                                                 "July",
                                                 "August"),
                              "summer",
                              ifelse(data3$month %in%
                                       c("September",
                                         "October",
                                         "November"),
                                     "fall", "winter")))

data3$season <- as.factor(data3$season)
namesDat3 <- data3[,c(2,4,19,22,23,24)]
namesDat3$month <- as.factor(namesDat3$month)
namesDat3$carrier <- as.factor(namesDat3$carrier)
namesDat3$week <- as.factor(namesDat3$week)
namesDat3$weekend <- as.factor(namesDat3$weekend)
namesDat4 <- namesDat3[,c(2, 4, 5, 6)]


#Flights Dataset Analysis
flights$hour <- ifelse(flights$hour == 24, 0, flights$hour)
flights_weather <- left_join(flights, weather)
flights_weather$total <- flights_weather$dep_delay +
  flights_weather$arr_delay
flights_weather2 <- filter(flights_weather, total > 0)

DatNew <- flights_weather2
DatNew$hour <- hour(as.POSIXct(DatNew$time_hour))
DatNew$week <- weekdays(as.Date(DatNew$time_hour))
DatNew$hour <- as.numeric(DatNew$hour)
DatNew$hour <- as.factor(DatNew$hour)
DatNew$weekend<- ifelse(DatNew$week %in%
      c("Saturday", "Sunday"), "weekend","weekday")
DatNew$month <- as.factor(DatNew$month)
DatNew$month <- plyr::mapvalues(DatNew$month,
  from = c("1", "2", "3", "4", "5", "6", "7", "8",
           "9", "10", "11", "12"),
                                to = c("January",
                                       "February",
                                       "March",
                                       "April",
                                       "May",
                                       "June",
                                       "July",
                                       "August",
                                       "September",
                                       "October",
                                       "November",
                                       "December"))
DatNew$season <- ifelse(DatNew$month %in% c("March",
                                            "April",
                                            "May"), "spring",
                        ifelse(DatNew$month %in% c("June",
                                                   "July",
                                                   "August"),
                               "summer",
                               ifelse(DatNew$month %in%
                                        c("September",
                                          "October",
                                          "November"),
                                      "fall", "winter")))

flights_weather2 <- DatNew
flights_weather3 <- flights_weather2[,c(20, 21,
                                        22, 23, 24,
                                        25, 26, 27,
                                        28)]



ui <- navbarPage("Flights Analysis",
                 tabPanel("Graphical",
                          sidebarLayout(
                            sidebarPanel(
                              actionButton("go",
                            "Graphical Flights Comparison"),

                              selectInput("response",
                                "Choose a response predictor:",
                              choices = names(namesDat3)),

                              selectInput("yearInitial",
                              "Choose year range (initial):",
                            choices = sort(unique(data3$year))),
                              selectInput("yearEnd",
                                "Choose year range (end):",
                              choices = sort(unique(data3$year))),
                              strong("Choose a
response predictor to visualize grouped changes in
mean departure delay over time from 2008 to 2016.")
                            ),
                            mainPanel(
                              plotOutput('plot', height = "900px")
                            )
                          )
                 ),
                 tabPanel("Table Summary",
                          sidebarLayout(
                            sidebarPanel(
                              actionButton("go1",
                              "Airport Flights Data"),

                              actionButton("go2",
                              "State Flights Data"),


                              selectInput("origin",
                              "Choose a origin airport:",
              choices = sort(unique(data2$OriginAirport))),

    selectInput("destination", "Choose a destination airport:",
          choices = sort(unique(data2$DestAirport))),

      selectInput("originState", "Choose a origin state:",
          choices = sort(unique(data2$OriginFState))),

  selectInput("destState", "Choose a destination state:",
          choices = sort(unique(data2$DestFState))),
  strong("Choose a origin and destination airport to
see the mean departure delay in the data table.
Alternatively, origin and destination states can
also be specified.")),
      mainPanel(
    dataTableOutput("view"),
    dataTableOutput("view2")
    )
      )
       ),
  tabPanel("2013 Weather & Flights Analysis",
  sidebarLayout(
      sidebarPanel(
    actionButton("go3", "Weather and Flight Delays"),

   selectInput("response2", "Choose a response predictor:",
          choices = names(namesDat4)),

          selectInput("weatherDis", "Choose a weather phenomena:",
          choices = names(flights_weather3)),
strong("Choose a response predictor and weather
occurrence from the dropdown to visualize changes
in total delay (departure delay plus arrival delay)
by the response variable chosen grouped by
weather event for LaGuardia, John F. Kennedy and
Newark Liberty International Airport in 2013.")
                            ),
              mainPanel(
              plotOutput('plot2', height = "900px")
                            )
                          )
                 ),
                 tabPanel("Flights Network Analysis",
                          sidebarLayout(
                            sidebarPanel(
                              actionButton("go5",
                              "Sample 500 Flights"),
                              strong("Click the button above to
generate a network of 500 flights visualizing departure delay
greater than 90 minutes. The network shows flights by airports
(vertices). The width of the edge segment shows the extent of
departure delay between origin and destined flights. Airport
can also be selected from the dropdown list (id). A data table
is also presented which lists departure delay for the queried
origin and destination airports.")
                            ),
                            mainPanel(
                              visNetworkOutput("network",
                                               height = "400px"),
                              #plotlyOutput('plot4',
                              height = "900px"),
                              dataTableOutput("view3")
                            ))
                 ),
                 tabPanel("Maps & Flights Analysis",
                          sidebarLayout(
                            sidebarPanel(
                              strong("The map shows the average
departure delay for United States airports from 2008 to 2016.
Hovering over the point will reveal
information about the average departure delay for the
                                state and airport selected.")
                            ),
                            mainPanel(
                              plotlyOutput('plot3')
                            )
                          )
                 )
)


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

  df_subset <- eventReactive(input$go1, {
    a <- data2 %>% filter(OriginAirport == input$origin & DestAirport
                          == input$destination)
    return(a)
  })

  df_subsetVis <- eventReactive(input$go5, {
    dat1 <- FinalDat[sample(nrow(FinalDat), 500, replace = FALSE,
                            prob = NULL),]
    dat2 <- dat1[,c(1, 13, 14, 6)]
    dat3 <- suppressWarnings(inner_join(dat2, airportsVis,
                                by = c("origin" = "iata")))
    dat4 <- suppressWarnings(inner_join(dat3, airportsVis,
                                by = c("dest" = "iata")))
    dat5 <- dat4[,c(5, 11, 4)]
    dat5$airport.x <- sort(dat5$airport.x)
    dat5$airport.y <- sort(dat5$airport.y)
    dat5 <- plyr::rename(dat5, c("airport.x" = "OriginAirport"))
    dat5 <- plyr::rename(dat5, c("airport.y" = "DestinationAirport"))
    dat5 <- plyr::rename(dat5, c("dep_delay" = "DepartureDelay"))
    return(dat5)
  })


  df_subset2 <- eventReactive(input$go2, {
```

```
  b <- data2 %>% filter(OriginFState == input$originState
                        & DestFState
                        == input$destState) %>%
    arrange(desc(meanDelay))
  return(b)
})

df_weekend <- eventReactive(input$go, {
  DatCarrier <- data3 %>% filter(year >= input$yearInitial
                                 & year <= input$yearEnd) %>%
    mutate(yearF = as.factor(year)) %>%
    group_by_("yearF", input$response) %>%
    summarise(MeanDep = mean(dep_delay))
  return(DatCarrier)
})

df_weather <- eventReactive(input$go3, {
  return(flights_weather2)
})

output$plot <- renderPlot({
  dfweek <-  df_weekend()
  p = ggplot(dfweek, aes_string(x = "yearF", y = "MeanDep",
                                group = input$response,
                                color = input$response)) +
    geom_point() + geom_line() +
    ggtitle(paste("Mean departure delay overtime by",
                  input$response))
  print(p)
  #ggplotly(p)
})

output$plot2 <- renderPlot({
  dfweek2 <-  df_weather()
  p1 = ggplot(dfweek2, aes_string(x = input$weatherDis, y = "total",
                                  group = input$response2,
                                  color = input$response2)) +
    geom_smooth()  + ggtitle(paste("Total Delay in" ,
                                   input$weatherDis, "in 2013"))
  print(p1)
  #ggplotly(p1) alternatively use interactive plotting
})

output$plot3 <- renderPlotly({
    g <- list(
      scope = 'usa',
      projection = list(type = 'albers usa'),
      showland = TRUE,
      landcolor = toRGB("gray85"),
      subunitwidth = 1,
      countrywidth = 1,
      subunitcolor = toRGB("white"),
      countrycolor = toRGB("white")
    )
    USAirSum3$MeanDepDelay <- round(USAirSum3$MeanDepDelay, 2)
    USAirSum3$OriginFState <- (tools::toTitleCase(USAirSum3$
                                                  OriginFState))
    p <- plot_geo(USAirSum3, locationmode = 'USA-states',
                  sizes = c(1, 250)) %>%
      add_markers(
        x = ~longitude, y = ~latitude, color = ~MeanDepDelay,
        hoverinfo = "text",alpha= 0.8,
        text = ~paste(OriginFState, "<br />", MeanDepDelay,
                      "<br />", OriginAirport)
      ) %>%
      layout(title = 'Mean Departure Delay by Airport from 2008-2016',
             geo = g)
    ggplotly(p)

  })

  output$network <- renderVisNetwork({
    dfweek2 <-  df_subsetVis()
    e2 <- dfweek2
    g2=graph.data.frame(e2)
    E(g2)$width <- dfweek2$DepartureDelay/7
    visIgraph(g2) %>% visOptions(highlightNearest = TRUE,
                                 nodesIdSelection = TRUE)
  })
  output$view <- renderDataTable(df_subset())

  output$view2 <- renderDataTable(df_subset2())

  output$view3 <- renderDataTable((df_subsetVis()))

})

options(shiny.sanitize.errors = TRUE)
```

```r
shinyApp(ui = ui, server = server)
```

Web Scraping for Airport and State Names

```r
library(rvest)
lego_movie1 <- read_html("http://www.airportcodes.us/us-airports-
                         by-state.htm")
dat2 <- html_nodes(lego_movie, ".c td")
airportText <- html_text(dat2)
airportText2 <- as.data.frame(airportText)
airportText2$airportText <- as.character(airportText2$airportText)

array2 = list()
for (i in 1:nrow(airportText2))
{
  if (nchar(airportText2[i,]$airportText) == 2)
  {
    array2 = c(array2, i)
  }
}
num <- c(1:2905)
array3 <- unlist(array2)
states2 <- data.frame(code=airportText2[array3,])

array4 = list()
for (i in 1:nrow(airportText2))
{
  if (nchar(airportText2[i,]$airportText) == 3)
  {
    array4 = c(array4, i)
  }
}
array4 <- unlist(array4)
counties <- data.frame(code=airportText2[array4,])
diffStates <- setdiff(num, array3)
diffStates2 <- setdiff(diffStates, array4)

#get airports (3 letter code + 1 indices)
code.numAir <- counties$code.num+1
code.numStates <- counties$code.num-2
airports <- data.frame(code=airportText2[code.numAir,])
states2 <- data.frame(code=airportText2[code.numStates,])
dataAir <- cbind(states2, counties, airports)
```

```r
AirData <- dataAir[,c(1, 3, 5)]
ld <- load("GroupedDest.Rda")
head(GroupedDest)
Origin <- GroupedDest$origin
Origin <- as.data.frame(Origin)
Origin$Origin <- as.character(Origin$Origin)
OriginMerge <- inner_join(Origin, AirData,
      by = c("Origin" = "code.airportText.1"))
OriginMerge #includes the airport

Dest <- GroupedDest$dest
Dest <- as.data.frame(Dest)
Dest$Dest <- as.character(Dest$Dest)
DestMerge <- inner_join(Dest, AirData,
      by = c("Dest" = "code.airportText.1"))
save(OriginMerge, file = "OriginMergeData.Rda")
save(DestMerge, file = "DestMergeData.Rda")

orig <- load("OriginMergeData.Rda")
dest <- load("DestMergeData.Rda")
group <- load("GroupedDest.Rda")

library(dplyr)
OriginMerge <- rename(OriginMerge, OriginState = code.airportText)
OriginMerge <- rename(OriginMerge, OriginAirport = code.airportText.2)

DestMerge <- rename(DestMerge, DestState = code.airportText)
DestMerge <- rename(DestMerge, DestAirport = code.airportText.2)
head(GroupedDest)

OriginJoin <- inner_join(OriginMerge, GroupedDest,
                    by = c("Origin" = "origin"))
OriginJoin2 <- inner_join(OriginJoin, DestMerge,
                    by = c("dest" = "Dest"))
unOrigin <- unique(OriginJoin2)
save(unOrigin, file = "AirlinesFull.Rda")
air <- load("AirlinesFull.Rda")
head(unOrigin)

library(rvest)
lego_movie2 <- read_html("http://www.50states.com/abbreviations.htm")
dat3 <- html_nodes(lego_movie2, "td")
airportText3 <- html_text(dat3)
airportText3 <- as.data.frame(airportText3)
```

```r
airportText3$airportText3 <- as.character(airportText3$airportText3)

numSt <- c(1:130)
statelist = list()
for (i in 1:nrow(airportText3))
{
  if (nchar(airportText3[i,]) == 2)
  {
    statelist = c(statelist, i)
  }
}
statelist <- unlist(statelist)
head(airportText3)
stateCode <- data.frame(code=airportText3[statelist,])
stateName <- setdiff(numSt, statelist)
stateName2 <- data.frame(code=airportText3[stateName, ])

statesName <- cbind(stateCode, stateName2)
statesToMerge <- statesName[1:50,]
head(statesToMerge)

statesToMerge$stateName <- statesToMerge$code
substates <- statesToMerge[,c(2,3)]
unOrigin$OriginState <- as.character(unOrigin$OriginState)
substates$stateName <- as.character(substates$stateName)
stateFin1 <- (unOrigin %>% inner_join(substates,
                  by = c("OriginState" = "stateName")))
substates <- rename(substates, DestName = stateName)
stateFin2 <- (stateFin1 %>% inner_join(substates,
                  by = c("DestState" = "DestName")))
head(stateFin2)
stateFin2 <- rename(stateFin2, OriginFState = code.x)
stateFin2 <- rename(stateFin2, DestFState = code.y)
save(stateFin2, file = "airportFullDat.Rda")
```

References

Ambati, S. (n.d.). Sparkling water = h2o + apache spark. databricks. Retrieved from `https://databricks.com/blog/2014/06/30/sparkling-water-h20-spark.html`

Borthakur, D. (n.d.). HDFS architecture guide. hadoop. Retrieved from `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`

Candel, A., Lanford, J., LeDell, E., Parmar, V., & Arora, A. (2015). Deep learning with h2o. Retrieved from `http://h2o-release.s3.amazonaws.com/h2o/rel-slater/9/docs-website/h2o-docs/booklets/DeepLearning_Vignette.pdf`

Checkpoint. (n.d.). H2O. Retrieved from `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/checkpoint.html`

Cook, D. (2016). *Practical machine learning with h2o*. O'Reilly Media.

Download apache spark. (n.d.). Apache Spark. Retrieved from `http://spark.apache.org/downloads.html`

Download h2o 3.10.0.6. (n.d.). H2O. Retrieved from `http://h2o-release.s3.amazonaws.com/h2o/rel-turing/6/index.html#R`

Download sparkling water 1.6.8. (n.d.). H2O. Retrieved from `http://h2o-release.s3.amazonaws.com/sparkling-water/rel-1.6/8/index.html`

H2O documentation r tutorial. (n.d.). Retrieved from `http://h2o-release.s3.amazonaws.com/h2o/rel-lambert/5/docs-website/Ruser/rtutorial.html`

Levin, A., & Sasso, M. (2016). The weather isn't the biggest cause of u.S. flight delays. Retrieved from `https://www.bloomberg.com/news/articles/2016-08-23/blame-the-airlines-not-the-weather-for-most-u-s-flight-delays`

Murthy, A. (2012, august). APACHE hadoop yarn – background and an overview. Retrieved from `https://hortonworks.com/blog/apache-hadoop-yarn-background-and-an-overview/`

Reddy, Y. B. (n.d.). Classification and automatic recognition of objects using h2o package. Retrieved from `http://www.gram.edu/offices/sponsoredprog/humangeo/`

docs/reddySPIE%202017-Related%20to%20BigData.pdf

Rickert, J. (2014). Diving into h2o. Retrieved from http://blog.revolutionanalytics.com/2014/04/a-dive-into-h2o.html

Spark programming guide. (n.d.). Apache Spark. Retrieved from http://spark.apache.org/docs/latest/programming-guide.html

Sparkling water. (n.d.). Retrieved from http://docs.h2o.ai/h2o/latest-stable/h2o-docs/faq.html#sparkling-water

Sparkling water (h20) machine learning. (n.d.). Retrieved from http://spark.rstudio.com/h2o.html

Sparklyr: R interface for apache spark. (n.d.). Retrieved from http://spark.rstudio.com/index.html

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. *Proceedings of the 30th International Conference on Machine Learning*. Retrieved from http://www.cs.toronto.edu/~hinton/absps/momentum.pdf