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

May 2017

Acknowledgements

| I would first like to thank the Amherst College Department of Mathematics and Statistics and Amherst College Department of Information Technology (IT) for facilitating this project. I would like to thank Professor Shu-Min Liao, Professor Pamela B. Matheson and Professor Amy Wagaman for reviewing the project as well as offering suggestions for improvement. Additionally I would like to thank Professor Eunice Kim and Professor Susan Wang for their assistance in the Statistics Program.

This project would not have been possible if it wasn't for Aaron Coburn's continuous assistance with the Hadoop platform. Additionally, I would like to thank Andy Anderson and Amherst College Department of IT for their assistance with the R server.

I would like to thank my classmates from Advanced Data Analysis, particularly Stephany-Flores Ramos, Jordan Browning and Levi Lee. I wouldn't have the necessary data wrangling and analytical skills if it wasn't for the insightful group work. Additionally I would like to thank Caleb Ki for collaborating on the Advanced Data Analysis group project.

Lastly I would like to thank Professor Nicholas Horton for all his support with Hadoop, the H2O machine learning platform and for stressing the importance of statistical analytics, git programming, and collaborative learning.

Table of Contents

List of Tables

List of Figures

Abstract

| This study aimed to predict departure delay from 2008 to 2016 against time assessing factors like year, carrier, air time, distance, week and season. Additionally this work provided an expository review of modern deep learning techniques like the H2O platform, which was used to perform modeling techniques like logistic regression and neural networks in R. Three different models were performed. A simple logistic regression was used to predict departure delay over 30 minutes against year, carrier, air time, distance, week and season. A subsequent simple logistic regression model was built to study influence of weather on predicting departure delay occurrence over 30 minutes against season, month, week, weekend, day, hour, distance and air time. Weather predictors included temperature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility. Lastly a deep learning neural network was built to study the influence of variables like year, month, carrier, distance, hour, week, weekend and season on predicting departure delay over 90 minutes. Additionally grid, random search and checkpoint variations of the original models were developed to facilitate hyperparameter specification. Overall, carrier type, season, distance and air time were most important at predicting departure delay occurrence over 30 minutes using logistic regression. Weather was not very important in predicting departure delay occurrence over 30 minutes. Hour (5, 6 and 9 am) and carrier status (Hawaiian Airlines, Northwest, Skywest and US Airlines) were most important at predicting departure delay over 90 minutes in the deep learning model.

Chapter 1

Introduction

1.1 Motivation

Airline-related delays (late arrival of 15 minutes) totaled about 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 a result of undetermined causes like a previously delayed flight. In total, about 1 million and 283 thousand hours of delay occurred in 2015 (Levin & Sasso, 2016).

This study strived to assess departure delay from 2008 to 2016. In addition, this work aimed to provide an expository review of H2O, the Deep Learning and Machine Learning platform, in predicting departure delay.

Chapter 2

Big Data Infrastructure

2.1 H2O

H2O is a big-data machine learning and predictive analytics platform, reputable for its fast and scalable deep learning capabilities. Machine learning algorithms include supervised and unsupervised learning algorithms like neural networks, tree ensembling, generalized linear regression models and k-means clustering. H2O provides deep learning capabilities through algorithms like perceptrons and feed forward neural networks. This platform is built as a Java Virtual Machine (JVM), which is an abstract computing environment for running a Java instance (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). This contained environment is optimal for performing distributed and parallel machine learning computations simultaneously on clusters.

2.1.1 H2O Installation Process

H2O installation entails loading Spark 1.6 and H2O version 3.10.06, both of which integrate with rsparkling. H2O package can be downloaded from (`http://h2o-release.s3.amazonaws.com/h2o/rel-turing/6/index.html#R`)[1]. The rsparkling package can be installed using devtools.

Since H2O connection necessitates initialization of a local Spark connection, same versions of Spark and Sparkling Water need be installed. 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].

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

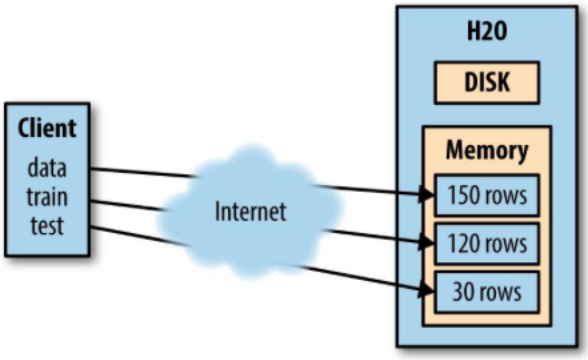

Figure 2.1: Client and H2O Cluster

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.).

In R, the sparklyr package provides an R interface for Apache Spark. This interface facilitates access to 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 for machine learning and deep learning algorithm implementation.

2.4 RSparkling

The rsparkling package facilitates data transfer between Spark and H2O dataframes. Rsparkling also allows access to Sparkling Water'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 in collaboration with node managers. The resource managers additionally contain a scheduler that arranges jobs based on resource requirements. Individual node managers are responsible for launching application containers and examining resource use (like memory and disk consumption). Updates are then reported back to the resource manager. Per-node application master negotiate resource containers with scheduler (Murthy, 2012).

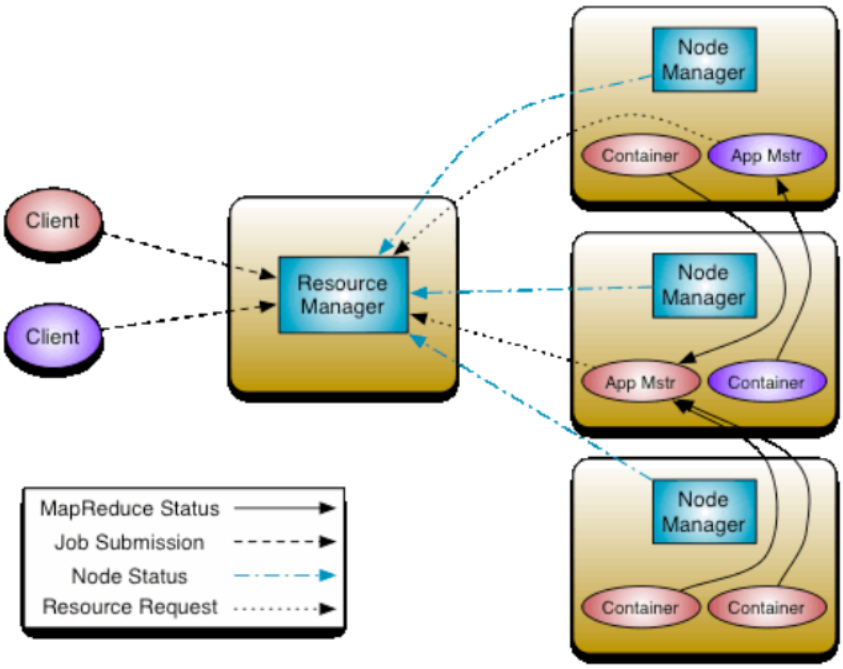

Figure 2.2: Hadoop YARN architecture

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 Application

3.1 Introduction

Flights dataset from The United States Department of Transportation Bureau of Transportation Statistics was used for Shiny visualization. The dataset (from 2008-2016) was set up in YARN-client cluster in the Hadoop server. Initially it contained variables like year, month and day of the trip, departure delay, arrival delay, carrier, tail number, distance covered, flight number, flight origin, destination and scheduled flight time. Additional predictors were created to assess departure delay. These variables included day of week, season, weekend status, and hour of flight delay.

Since only flights with departure delay greater than 90 minutes were used for deep learning analysis, a table was constructed to show the percentage of flights with departure delay greater than 90 minutes from the orignal dataset (created as a combination of data from individual years). A random sample of 200,000 observations with departure delay greater than 90 minutes was selected from the observations shown in table 3.1.

Table 3.1: Percentage of observations with >90 minutes delay

Year	TotalFlights	NumberDelayed	PercentDelayed
2008	7009726	235476	3.36
2009	6450285	176710	2.74
2010	6450117	176460	2.74
2011	6085281	176120	2.89
2012	6096761	166902	2.74
2013	6369482	208306	3.27
2014	5819811	198631	3.41
2015	5819079	188433	3.24
2016	2289826	63429	2.77

3.2 Web Scraping

Since carrier, departure airport 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 the 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

A shiny application was created for initial exploratory analysis. Graphical, tabular and weather analysis was performed. Images from the shiny app are shown below each commentary. The app itself can be accessed at `https://aj17.shinyapps.io/flightsapp/`.

3.3.1 Tabular Analysis

The tabular analysis displayed 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.

3.3.2 Path Analysis

Path analysis tab shows flights from a specified origin and destination state for a chosen year for all flights with departure delay greater than 90 minutes. Point width represents the extent of departure delay in minutes. The application was set up so that a point click reveals information about the orgin and destination state and airports along with the average departure delay for that trip in minutes. An example of paths from California to New York is shown by `ref("shiny90")`

3.3.3 Departure Delay Analysis

Delay by State, Region and Division

Departure delay analysis was graphically performed at state, region and division level. The first tab shows departure delay by airports for the selected state and the aggregated delays for the selected state over all airports from 2008 to 2016. In addition, departure delay by four regions (Midwest, Northeast, South and West) was analyzed. Each region is further analyzed by division. Midwest region is analyzed by East North Central and West North Central divisions. Northeast is analyzed by Middle Atlantic and New England divisions. South by East South Central, West South Central and South Atlantic divisions. Lastly Pacific and Mountain divisions are analyzed for Western United States. For each region analysis, state-wide trends in

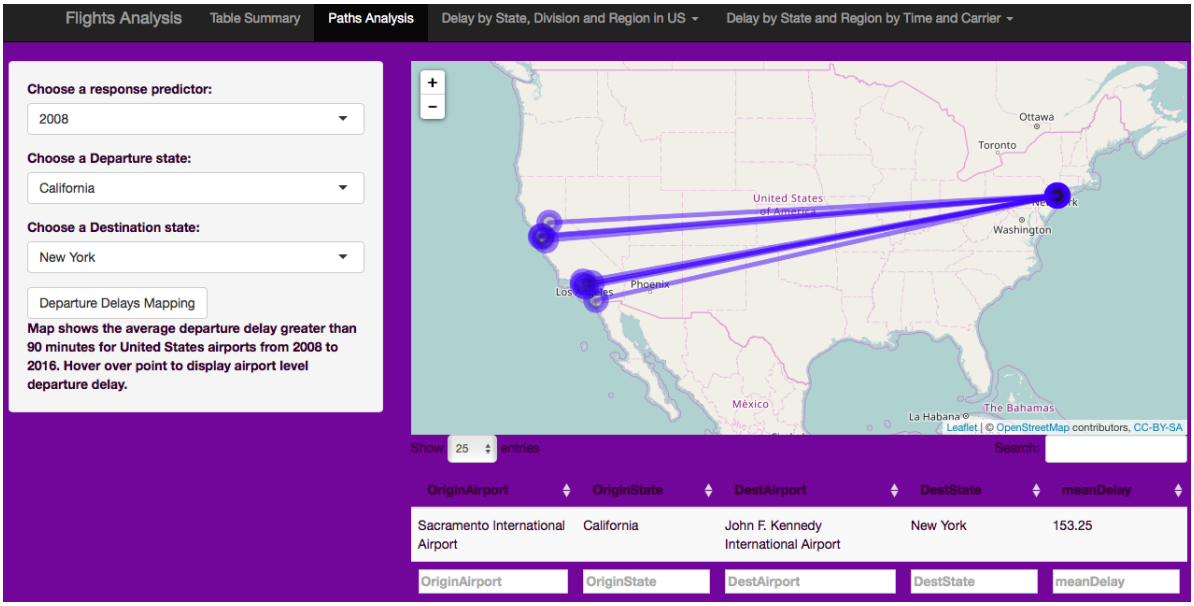

Figure 3.1: Paths Analysis from California to New York in 2008

the divisions for that region as well as aggregate departure delay over 2008 to 2016 are shown. An output example for Northeast is shown by Figure 3.2.

Delay by State and Region over Time and Carrier

In addition to departure delay analysis at the state, region and division level, delay was analyzed by variables including hour, day of week, weekend status, month, season and carrier type. Analysis was performed at regional (Midwest, notheast, south, west) and state levels. A general trend pointed towards high delays in South and Midwest in late night/early morning hours. Northeast region appears to have the highest delays over the week with the highest delays occurring on Sunday and weekends. Departure delays in Northeast (June, July, August) are high in the summer season while delays are high in spring (April, May) in South. In regards to carrier analysis, Hawaiian Airlines experiences the highest delay in Northeast and West region. Example below shows delay by weekday for Midwest, Northeast, South and Western regions as depicted by Figure 3.3.

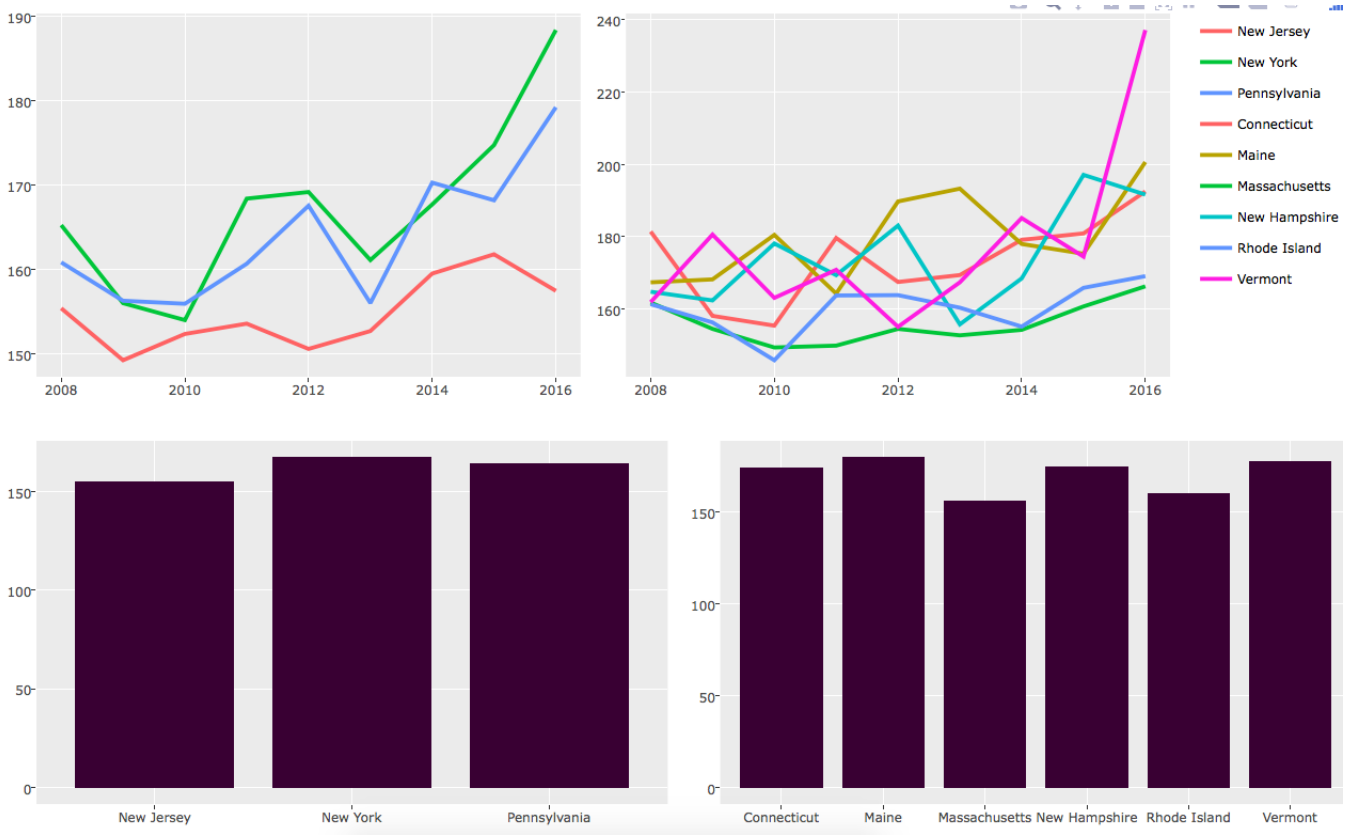

Figure 3.2: Departure Delay by Region and Division

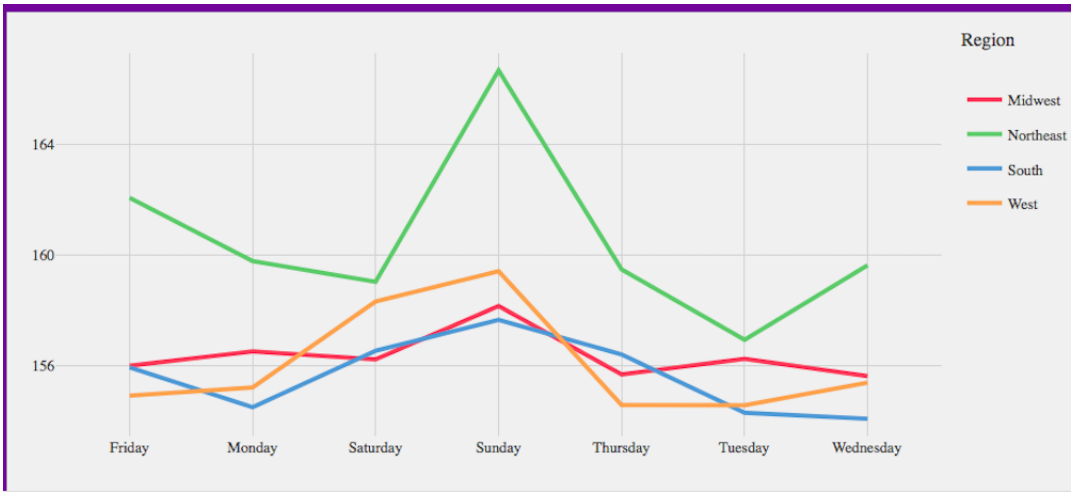

Figure 3.3: Departure Delay by Region over Weekday

3.3.4 Aggregate United States Departure Delays

The last shiny panel displays the mean departure delay by airport from 2008 to 2016 in the United States for all flights with departure delay greater than 90 minutes. The dataset for this panel was created by filtering all flights with delay greater than 90 minutes for every year from 2008 to 2016, resulting in 1,590,467 observations. This data was then aggregated by origin and destination airports and year to produce a smaller dataset containing 39,277 observations. Hovering over the points displays the state, airport and the average departure delay for that airport.

Chapter 4

Logistic Regression

4.1 Introduction

H2O platform was used to construct a logistic regressions model to predict departure delay occurrence over 30 minutes for flights data from The United States Department of Transportation Bureau of Transportation Statistics from 2008 to 2016. Cutoff of 30 minutes provided adequate sample size for flights that either experienced or did not experience departure delay over 30 minutes. Analysis was performed on a random sample of 200,000 observations since this dataset of this size could easily be transferred to the Spark environment. Occurrence of delay over 30 minutes was assessed against year, carrier, air time, distance, week and season as predictors. Initially, hour, month and weekend status predictors were also used to estimate departure delay incidence though they appeared to have no importance and therefore were not used in subsequent analyses.

4.2 H2O Connection

H2O connection was established to local host using h2o.init() using default 1GB of memory. Additional cluster memory can be allocated with max_mem_size specification. Since this project was performed using the Apache Spark platform, spark connection was established with the YARN Hadoop cluster.

```
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
kinit()
```

```
klist

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

4.3 Spark Data Integration

Once spark connection was estabished, flights dataset from 2008-2016 was allocated by copying flights data from 2008 to 2016 from Hadoop to the Spark environment (see Appendix 2 for full details on the dataset creation).

```
#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 data was copied, it was partitioned into test and training sets. An approximate 75/25 partition was used where the training dataset was allocated about 75% of the data (149,789 observations) and test set was allocated about 25% (50,211 observations).

```
#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

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

4.6 Modeling

Once data was partitioned, logistic regression model was specified as shown. The h2o.glm function was used to specify a binomial family function. Additional arguments in the h2o.glm function included 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 specified lambda space). H2O models can additionally be stopped early with specification of 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, the logistic model shown below attempted to predict departure delay occurrence by variables like year, 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 0.0088 and the AUC was 0.586 as shown by Figure 4.1. The R^2 value indicated that only about 0.9% of the variation in departure delay was accounted for by predictors year, carrier, air time, distance, week and season. In addition to the h2o.performance function, h2o.auc and h2o.confusionMatrix (see Figure 4.2) were

used to retrieve the analogous parameters. The AUC curve was visualized as shown by figure Figure 4.3 with the plot(h2o.performance) command. The curve with arched midway visually shows area convergence of about 50%.

```
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 da  h2o.confusionMatrix(object, ...)

MSE:  0.09265474
RMSE:  0.3043924
LogLoss:  0.3300536
Mean Per-Class Error:  0.4368844
AUC:  0.5864663
Gini:  0.1729325
R^2:  0.008824555
Null Deviance:  100235.1
Residual Deviance:  98876.81
AIC:  98956.81

Confusion Matrix for F1-optimal threshold:
          No   Yes    Error           Rate
No     65091 69064 0.514807  =69064/134155
Yes     5612 10022 0.358961    =5612/15634
Totals 70703 79086 0.498541  =74676/149789

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold    value idx
1                       max f1  0.102764 0.211613 234
2                       max f2  0.072754 0.375042 317
3                 max f0point5  0.129113 0.162729 139
4                 max accuracy  0.235908 0.895620   0
5                max precision  0.235908 0.333333   0
6                   max recall  0.018110 1.000000 398
7              max specificity  0.235908 0.999985   0
8             max absolute_mcc  0.102461 0.077318 235
9   max min_per_class_accuracy  0.108277 0.558906 218
10 max mean_per_class_accuracy  0.102764 0.563116 234
```

Figure 4.1: Model Performance

```
> h2o.confusionMatrix(regmod)
Confusion Matrix for max f1 @ threshold = 0.102764246813049:
          No   Yes    Error           Rate
No     65091 69064 0.514807  =69064/134155
Yes     5612 10022 0.358961    =5612/15634
Totals 70703 79086 0.498541  =74676/149789
```

Figure 4.2: Confusion Matrix

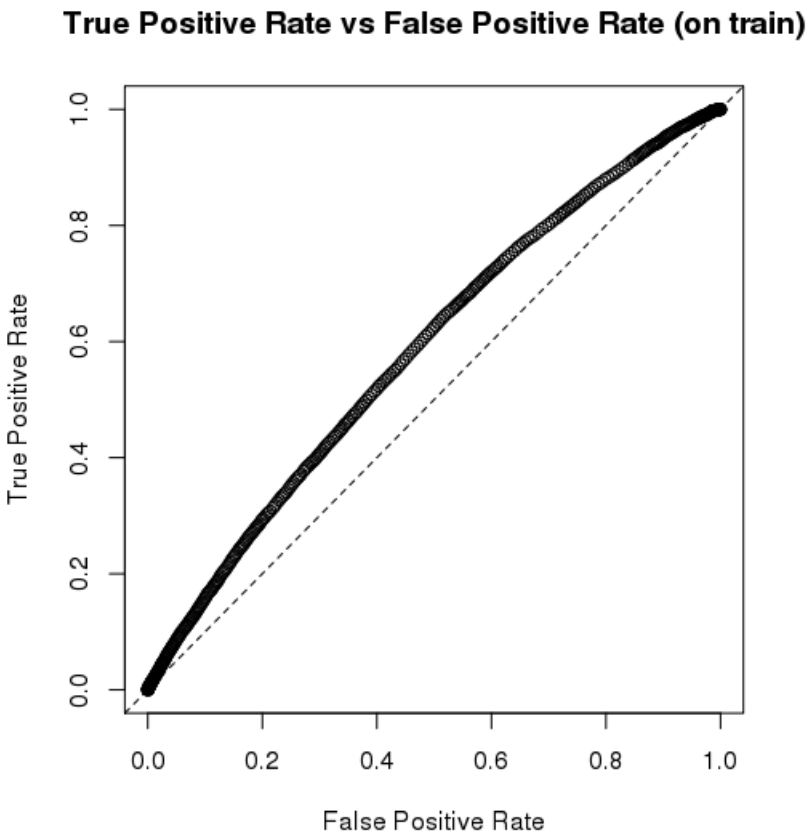

Figure 4.3: AUC Curve

As the variable importance plot in Figure 4.4 shows, carrier type was the most important predictor of departure delay over 30 minutes. Hawaiian Airlines (HA) was most negatively associated with departure delay. Carrier Spirit (NK) was positively associated with departure delay. Additionally, carriers Alaska (AS), US Air (US) were negative predictors of departure delay while carriers JetBlue (B6) and Atlantic Southeast Airlines were positive predictors of departure delay greater than 30 minutes. Season fall was a negative predictor of departure delay while summer was a positive predictor of departure delay over 30 minutes. Air time was a positive indicator of departure delay over 30 minutes while distance was a negative predictor of departure delay over 30 minutes.

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

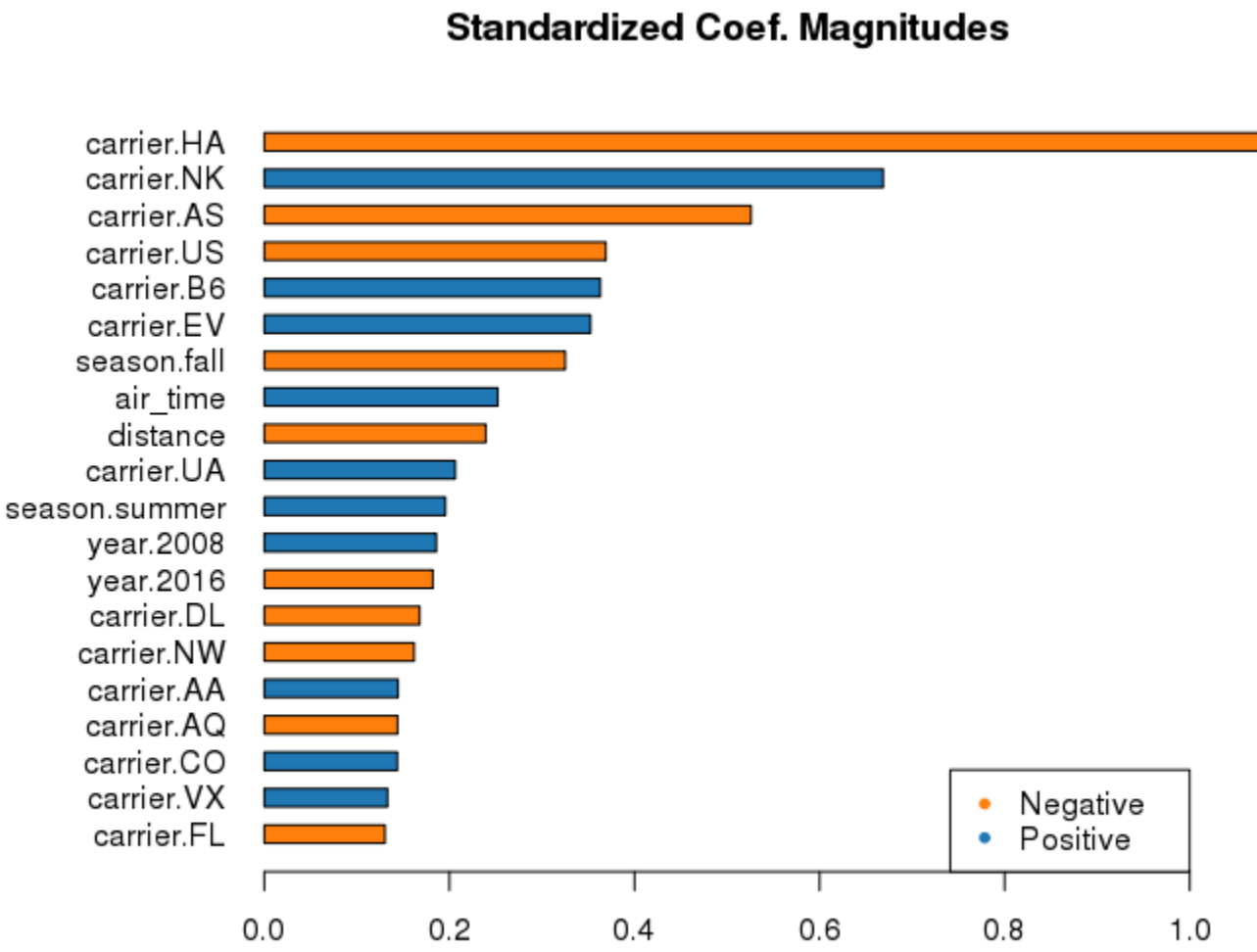

Figure 4.4: Variable Importance for Logistic Regression Analysis

4.8 Making Predictions

After model assessments were analyzed, predictions were performed on the test set. The accuracy of the test set was calculated and compared with the accuracy of the cross-validated training set. Accuracy of the test set of 0.5077 compared with the accuracy of the training data of 0.5015. While the accuracy of the training and test data was not very high, the error rates for both data were comparable thereby reducing overfitting likelihood.

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

4.9 Conclusion

This chapter discussed concepts like setting up a H2O Yarn connection, dataset partitioning, linear regression modeling and model assessments. Overall carrier type appeared to be most important in predicting departure delay over 30 minutes. Predictors like season, air time and distance were important. It must be noted though since the R^2 value was only about 0.0088 meaning about 0.9% of the variation in departure delay was covered by the chosen predictors, the model does not have strong practical significance or predictive ability given an accuracy of about 50%.

Chapter 5

Logistic Regression and Weather

5.1 Introduction

The H2O platform was used to perform logistic regression to predict departure delay over 30 minutes using weather data from the nycflights13 package for LaGuardia, John F. Kennedy and Newark Liberty International airports in 2013. Analysis was performed on data containing 48,126 observations. Occurrence of delay over 30 minutes was assessed against season, month, week, weekend, day, hour, distance and air time. Weather predictors included temperature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility.

5.2 Data Partitioning

Data from the nycflights13 package was copied in the Spark environment followed by a 75/25 partitioning. The training data contained about 36,153 observations and the test set contained 11,973 observations.

```
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

Linearity was assumed given the binary nature of the response variable assessing whether or not departure delay over 30 minutes occurred. Randomness and independence may not necessarily be valid assumptions since a late flight typically results in subsequent delays. The study proceeded with caution.

5.4 Modeling

Logistic model was used to predict departure delay against weather predictors like season, month, week, weekend, day, hour, distance, air time, temprature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility. Same parameters as specified in 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 the h2o.performance function. R^2 for this model was 0.109 and the AUC was 0.704 as shown by Figure 5.1. The R^2 value indicated that about 11% of the variation in the departure delay variable was captured by season, month, week, weekend, day, hour, distance, air time, temprature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility. 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.

```
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:  0.1853124
RMSE:  0.4304793
LogLoss:  0.5508155
Mean Per-Class Error:  0.3536385
AUC:  0.7043045
Gini:  0.408609
R^2:  0.1087181
Null Deviance:  43849.56
Residual Deviance:  39827.27
AIC:  39999.27
```

Figure 5.1: Model Performance

```
> h2o.confusionMatrix(regmodWeather)
Confusion Matrix for max f1 @ threshold = 0.265641713794513:
          No   Yes    Error          Rate
No     14070 11423 0.448084  =11423/25493
Yes     2763  7897 0.259193   =2763/10660
Totals 16833 19320 0.392388  =14186/36153
```

Figure 5.2: Confusion Matrix

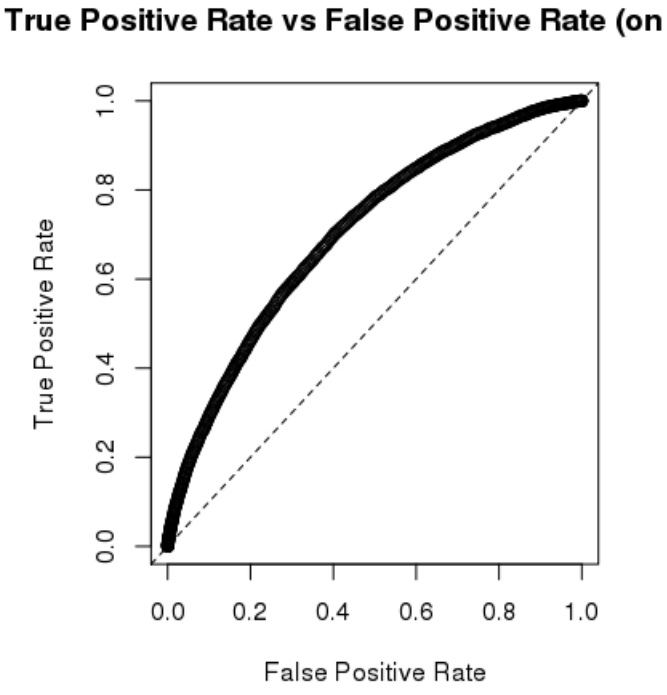

Figure 5.3: AUC Curve

As shown in the variable importance plot in Figure 5.4, air time and distance were important in predicting departure delay over 30 minutes. While air time was a negative predictor of departure delay, distance was a positive indicator of departure delay over 30 minutes. Hours 5, 6 and 7 am were negative predictors of departure delay over 30 minutes. In comparison, hours 6, 7 and 8 pm were positive predictors of departure delay over 30 minutes. Intuitively, this result suggested that there is a higher likelihood of experiencing departure delay greater than 30 minutes in evening than in early morning hours. Weather was not very important since humidity was the only important positive predictor (ranked 36th out of 40) of departure delay greater than 30 minutes.

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

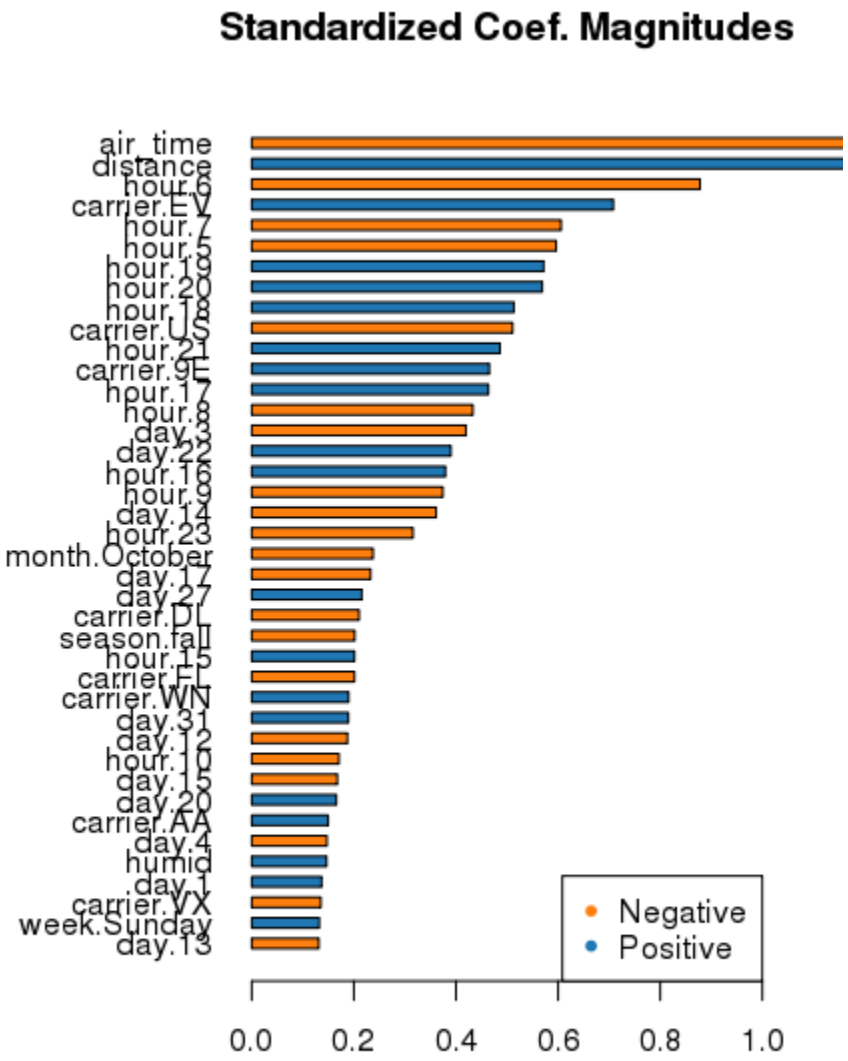

Figure 5.4: Variable Importance for Weather Logistic Regression

5.6 Making Predictions

After model assessments were analyzed, predictions were 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 0.612 compared with the accuracy of the training set of 0.608. Since the error rates were similar, overfitting occurrence was reduced.

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

5.7 Conclusion

This chapter assessed the effects of weather on departure delay aiming to examine the role of weather metrics like temperature, dewpoint, humidity, wind direction, wind speed, wind gust, precipitation, pressure and visibility on departure delay while controlling for predictors like distance and air time. Overall air time and distance were the most important predictors. Hour was also important in predicting departure delay over 30 minutes. Predictors assessing weather were not important. While the model including weather produced a R^2 of 0.10 (higher than the R^2 for logistic model without weather), R^2 of 0.10 nonetheless indicates that most of the variation in departure delay is unaccounted for by the explanatory predictors including weather.

Chapter 6

Deep Learning

6.1 Introduction

6.1.1 Machine Learning and Deep Learning

Machine learning is the process of insight extraction from natural data like speech, text and images to develop pattern-recognition systems, transcribe speech in natural language processing (NLP), develop reinforcement learning models, identify data anomalies and create user-targetted recommendation systems. Conventional machine learning models are limited in their capacity to build intricate models from raw data of large scale (dataset like ImageNet containing about 15 million images) (Krizhevsky, Sutskever, & Hinton, n.d.). Representation learning methods like deep learning provide the ability to automatically extract data insights for detection or classification. Deep learning provides multiple levels of representation through its non-linear module functionality, which transforms data from initial level to a more complex state (LeCun, Bengio, & Hinton, 2015). Learned process' ability to automatically extract insights using general-purpose learning process provides agility and dynamicity to deep learning models.

Machine Learning is conventionally divided in supervised and unsupervised learning. Supervised learning requires labeled data and is used to perform operations like image processing. Supervised learning includes algorithms like regression, neural networks, random forest and support vector machines. Unsupervised learning, on the other hand, attempts to locate patterns in unlabeled data and includes algorithms like k-nearest neighbor, which use distance metrics to find natural data groupings.

This section explores the application of neural networks in a Deep Learning context, which describe layered neural networks. Deep learning is used in fields like science, advertisement, and business for high dimensional natural data processing including image and speech recognition, drug molecule activity prediction and natural language processing procedures like sentiment analysis and topic classification.

6.2 H2O Deep Learning

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 hidden layers irrelevant. Examples of activation functions include sigmoid and rectified linear unit (ReLU). The multi-layer platform consists of layers of interconnected neurons, which are composed of nonlinear layers 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).

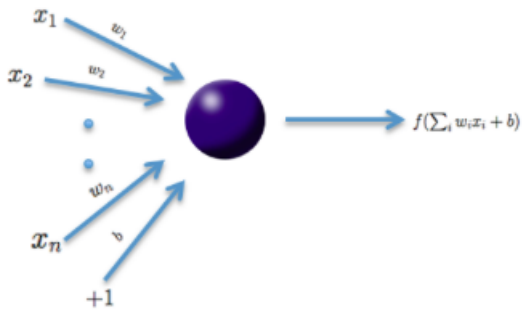

Figure 6.1: Neural Network

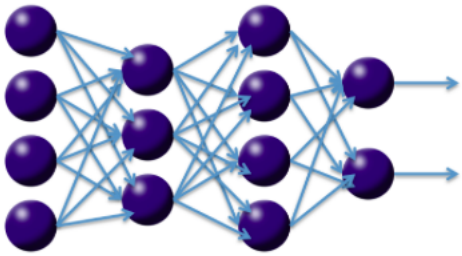

Figure 6.2: Hidden Layers

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 categorical and numerical data processing along with automatic missing data imputation.

H2O deep learning platform was used to predict departure delay over 90 minutes. The response variable was a continuous predictor capturing extent of departure delay

in minutes. While the dataset used to perform logistic regression contained a binary response indicating the occurrence of departure delay over 30 minutes, deep learning was performed with a stringent criteria of 90 minutes since a larger delay is more likely to incur financial expenses. The explanatory predictors of interest for the deep learning model included year, month, carrier, distance, hour, week, weekend and season.

6.3 Data Partitioning

Before data preparation and model building, connection was established to the YARN client. Following connection, data sample of 200,000 was obtained, which provided easy transferability to the Spark environment. Since hyperparameter optimization was performed in the deep learning algorithm, a validation data set in addition to a test set was used for additional verification. Data was split 60/20/20 consisting of 60% training, 20% validation and 20% test set.

```
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)
```

6.4 Model Building

Following data partitioning, h2o.deeplearning function was used to predict departure delay over 90 minutes as a function of year, month, 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 paramaters 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, respectively. 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. An epoch of one, indicating a single data iteration, was specified. In addition, 5-fold cross validation was used. Tanh activation layer was used since it is more adept at exponentially rising functions and consequently appropriate for regularization.

```
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 was 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, hour and carrier type were most important at predicting departure delay greater than 90 minutes. Hours 6 am and 5 am along with 9 pm were important at predicting departure delay greater than 90 minutes. Additionally carriers Northwest (NW), US Air (US) and Frontier (F9) were important at predicting departure delay greater than 90 minutes. Besides hour and carrier type, flights in April were important at predicting departure delay greater than 90 minutes.

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

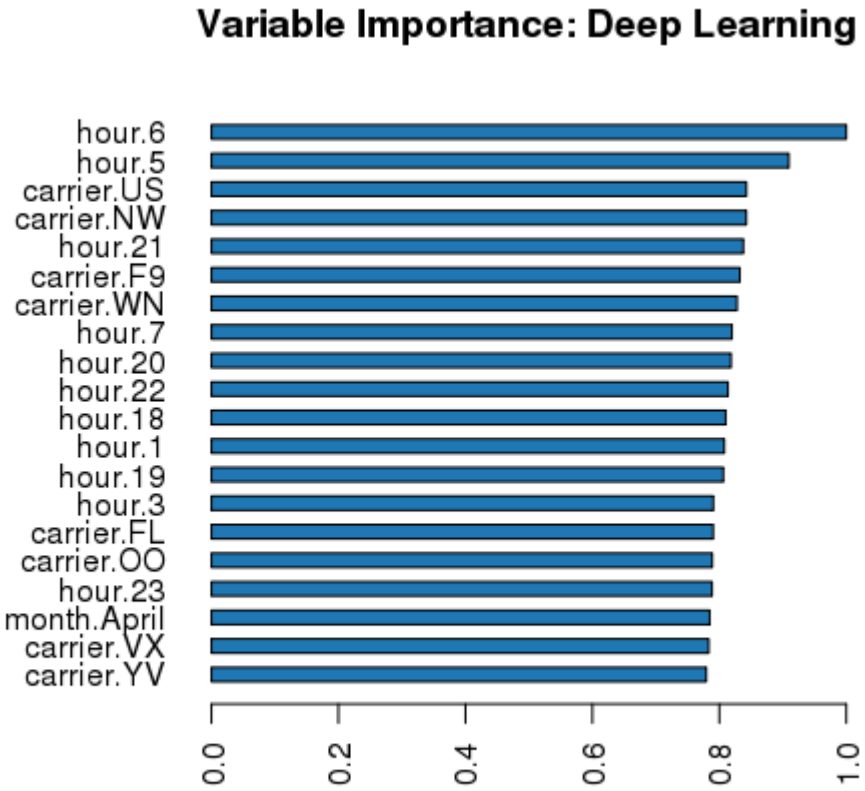

Figure 6.4: Deep Learning Variable Importance

6.5 Model Assessment

Since the response variable was a continuous indicator of departure delay, mean squared error (MSE) was used as an error metric to assess the performance of the deep learning model.

As shown by Figure 6.5, the MSE on training data was 6522.2 while MSE on validation data was 6918.5. In comparison, MSE on test data was 7010.6. Since MSE for the validation and test data was higher than MSE on training data, this was an indication of good fit. Overfitting did not appear to be an issue since the difference between the MSE of the training, validation and test set was not especially high. The model results however need to be considered cautiously due to the high MSE.

```
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 9978 samples **

MSE:  6522.245
RMSE:  80.76041
MAE:  52.66537
RMSLE:  0.3835295
Mean Residual Deviance :  6522.245

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

MSE:  6918.539
RMSE:  83.17776
MAE:  52.93873
RMSLE:  0.3846599
Mean Residual Deviance :  6918.539

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

MSE:  7010.565
RMSE:  83.72912
MAE:  52.8193
RMSLE:  0.3840508
Mean Residual Deviance :  7010.565
```

Figure 6.5: Deep Learning Model Performance

6.6 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")
```

6.7 Grid Search Model

H2O's search functionality facilitated experimentation with different hyperparameter combinations. All possible combinations of the hyperparameters were 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 adjusts 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)
)
```

Following hyperparameter specification, h2o.grid functionality was used for model iteration. In order to expedite the model building process, stopping metrics were specified so that 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 specified hyperparameters, epoch of 10 was chosen for model building. Momentum was specified to reduce algorithm halting at 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. Direct indexing in the grid object can be used to retrieve the optimal model by MSE along with associated parameters. The gridDeep grid search resulted in an optimal model with parameters including the Tanh activation layer, hidden layer of (20,20), input_dropout_ratio of 0 and rate of 0.03 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] 20 20
> optimal@allparameters$input_dropout_ratio
[1] 0
> optimal@allparameters$rate
[1] 0.03
```

Figure 6.6: Grid Model Parameters

6630.62 whereas MSE for the validation data was 6883.68 and 6972.59 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 validation and test data for grid search model (optimal) are lower than the MSE for the validation and test data for the original non-grid search model (deepmod) (validation MSE of 6918.5 and test MSE of 7010.6).

Variable importance plot was used to view the most important predictors. As Figure 6.8 shows, hour was most important in predicting departure delay greater than 90 minutes. Hour 6 and 5 am along with 9 pm were also important. Carriers Southwest (WN) and JetBlue (B6), weekday status of a weekend, season being summer and weekday being Friday were additionally important in predicting departure delay over 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 9931 samples **

MSE:  6630.617
RMSE:  81.4286
MAE:  51.79185
RMSLE:  0.3800612
Mean Residual Deviance :  6630.617

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

MSE:  6883.675
RMSE:  82.96792
MAE:  52.12437
RMSLE:  0.3808864
Mean Residual Deviance :  6883.675

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

MSE:  6972.585
RMSE:  83.502
MAE:  52.02304
RMSLE:  0.3801282
Mean Residual Deviance :  6972.585
```

Figure 6.7: Grid Model Performance

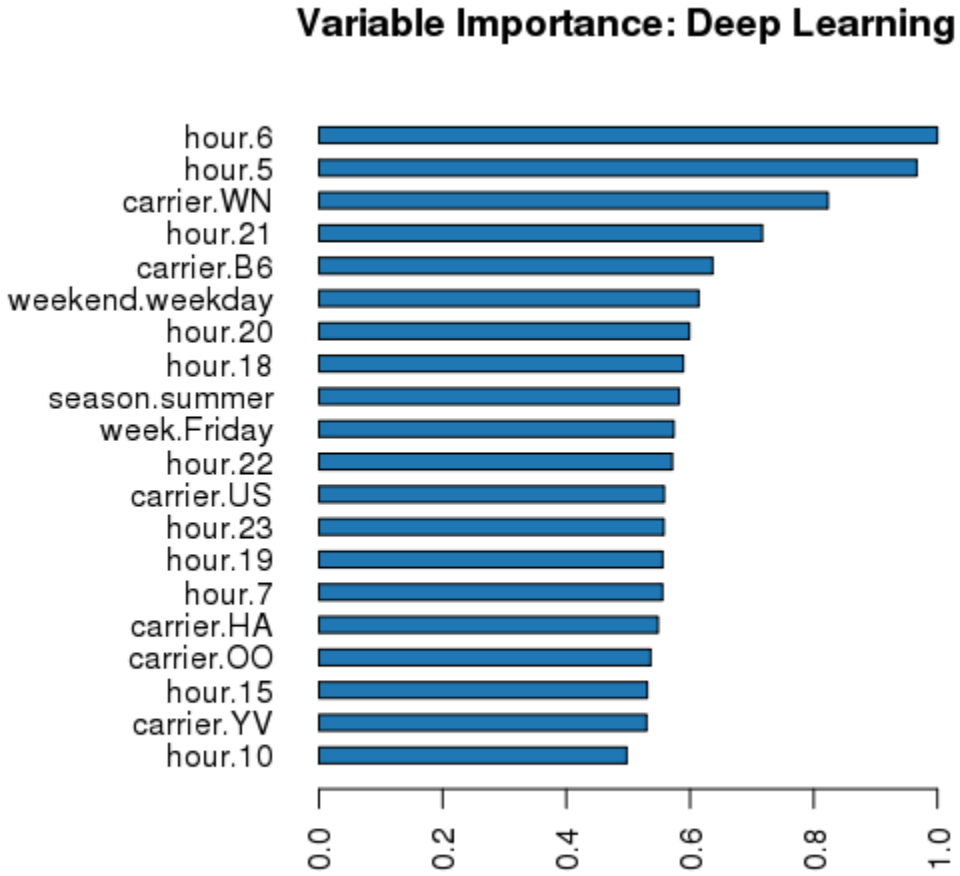

Figure 6.8: Grid Search Variable Importance

6.8 Random Grid Search Model

In comparison to grid search which iterated over parameter combinations exhaustively and sequentially, random grid search model was used to accelerate hyperparameter selection process. Random grid search proceeds to randomly search the user specified space based on 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 tested along with 0.0 and 0.02. In addition, various 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 was used to specify additional fixed parameters. Additional parameters included definition of epochs of 40, max_w2 of 10, score_validation_samples of 10000 and score_duty_cycles of 0.025. The score_validation_samples specified the number of validation set samples for scoring while the score_duty_cycles specified 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 did 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=40,
  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 random search models can be printed using for-loops. Additionally, the best model and its associated parameters can be viewed. As shown by Figure 6.9, optimal model generated by random grid search had an activation function of Tanh, hidden layer of (40, 40, 40), input_dropout_ratio of 0, rate of 0.03, l1 of 4.4e-05 and l2 of 7.4e-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,]
optimalRand <- h2o.getModel(grid@model_ids[[1]])

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

As Figure 6.10 indicates, the MSE on the optimal random model for the training data was 6560.97 whereas MSE for the validation set was 6577.3 and 6966.74 for the test set. The training, validation and test errors were lower than those yielded by the grid search model (6630.62 for training, 6883.68 for validation and 6972.59 for test). The difference between the MSE for the validation and test set raised concerns about overfitting. The errors for the random search model were lower than the initial deep learning model deepmod validation and test errors (validation MSE of 6918.5 and test MSE of 7010.6).

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, carrier Hawaiian Airlines (HA) were most important at predicting departure delay greater than 90 minutes, a result corroborated by shiny app. Additionally hour (5, 6 and 9

```
> optimalRand@allparameters$activation
[1] "Tanh"
> optimalRand@allparameters$hidden
[1] 40 40 40
> optimalRand@allparameters$input_dropout_ratio
[1] 0
> optimalRand@allparameters$rate
[1] 0.03
> optimalRand@allparameters$l1
[1] 4.4e-05
> optimalRand@allparameters$l2
[1] 7.4e-05
```

Figure 6.9: Random Grid Model Parameters

```
> h2o.performance(optimalRand, train = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10053 samples **

MSE:  6560.972
RMSE:  80.99983
MAE:  51.90851
RMSLE:  0.3783547
Mean Residual Deviance :  6560.972

> h2o.performance(optimalRand, valid = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on temporary validation frame with 10066 samples **

MSE:  6577.3
RMSE:  81.10056
MAE:  51.41054
RMSLE:  0.3769872
Mean Residual Deviance :  6577.3

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

MSE:  6966.744
RMSE:  83.46702
MAE:  52.01546
RMSLE:  0.3804574
Mean Residual Deviance :  6966.744
```

Figure 6.10: Random Grid Performance

am)and carrier (SkyWest Airlines (OO), Northwest Airlines (NW) and US Airways (US)) appeared to be important predictors of departure delay greater than 90 minutes.

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

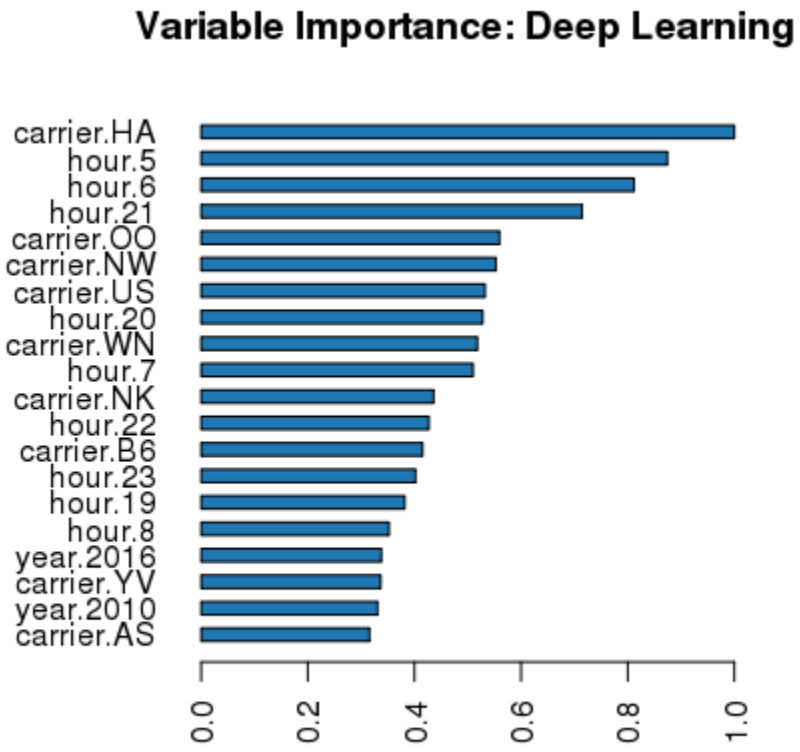

Figure 6.11: Random Grid Variable Importance

6.9 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].

With some reservations about overfitting in the random grid search model, higher l1 and l2 parameters were used to see if better performing model can be produced than the random grid search model. Additionally higher epochs (50) was specified. 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 (40, 40, 40) and rate (0.03) were used since these parameters can't be altered in the checkpoint model.

```
max_epochs <- 50
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(40, 40, 40),
  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 = 0.03,
  variable_importances=T
)
```

As shown by Figure 6.12, MSE on the training data was 6874.65, 6899.89 for validation and 6992.02 for test data. Though the test set MSE for the checkpoint

[1] ("Checkpoint," n.d.)

model is higher than the test MSE produced by the random grid model, the test MSE for the checkpoint model is closer to the validation set than the MSE for the validation and test sets in the model produced by random search grid criteria. The checkpoint model thus seems to have produced a better fitting model with reduced chance of overfitting than the random grid search model.

```
> h2o.performance(checkpoint, train = TRUE)
H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10027 samples **

MSE:  6874.653
RMSE:  82.91353
MAE:  50.85253
RMSLE:  0.3768464
Mean Residual Deviance :  6874.653

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

MSE:  6899.888
RMSE:  83.06557
MAE:  51.41236
RMSLE:  0.379712
Mean Residual Deviance :  6899.888

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

MSE:  6992.019
RMSE:  83.6183
MAE:  51.28932
RMSLE:  0.3788454
Mean Residual Deviance :  6992.019
```

Figure 6.12: Checkpoint Performance

The variable importance plot shown in Figure 6.13, is comparable to the variable importance plot produced by random grid search model. Hawaiian carrier, 5 am, 6 am and 9 pm appeared to be the most important predictors of departure delay greater than 90 minutes. Additionally SkyWest Airlines, Northwest Airlines and US Airways appeared to be next important in predicting departure delay greater than 90 minutes.

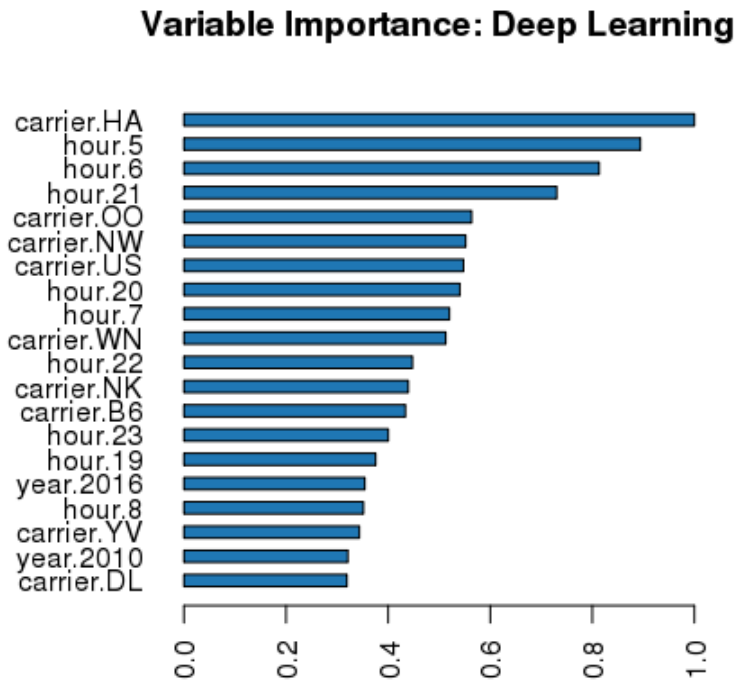

Figure 6.13: Checkpoint Variable Importance

6.10 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, hour and carrier status predictors were most important predictors of departure delay greater than 90 minutes. Hour 5 am, 6 am, 9 pm along with carriers Hawaiian Airlines, Northwest, Skywest, US Airlines, Southwest and JetBlue were also important.

In regards to model performance, the grid search and the random grid model both performed better than the initial deep learning model. In addition, the checkpoint model reduced overfitting, thereby producing better performance than the random grid model.

Conclusion

This project assessed departure delay from 2008 to 2016. It involved navigation of both the R and Hadoop servers and utilized packages like h2o and rsparkling amongst others. Shiny application was used for initial data visualization. H2O platform was used to perform logistic modeling and deep learning. Logistic regression was performed with binary departure delay indicator variable (categorized by whether or not departure delay was over 30 minutes). In comparison, deep learning was performed with a continuous indicator of departure delay with data filtered for all flights with departure delay greater than 90 minutes. As described in the Deep Learning chapter, criteria of 30 minutes was chosen for logistic regression since it provided an adequate number of observations that either experienced or did not experience departure delay higher than 30 minutes. In comparison, a more stringent criteria of 90 minutes was used to model departure delay in neural network model since the interest was in modeling higher departure delay, which typically results in greater inconvenience. Weather was also assessed via logistic regression. Conceptually, this project explored big data infrastructure of platforms like Hadoop and Apache Spark, algorithms like neural networks and model optimization through grid and random based hyperparameter specification.

6.11 Limitation

A limitation of this study was that although the original flights dataset had about a million observations, only a sample of about 200,000 observations was copied in the Spark environment. Capacity of Spark can be improved to handle additional data. Parallel computing through the parallel package can be used to split the original data in smaller datasets followed by a computation aggregation.

6.12 Future Work

Future extensions of this project can include building data pipelines to dynamically collect and study weather data for the associated flights. Data containing security alerts can also 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. Additionally, 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 manufacture information.

Appendix A

Appendix

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

```
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"
```

```
)
)
)

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)
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

```
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))
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)]

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(plyr)
library(dplyr)
library(mosaic)
library(base)
library(plotly)
library(ggplot2)
library(nycflights13)
library(lubridate)
library(igraph)
require(visNetwork)
library(gridExtra)
library(grid)
library(leaflet)
library(ggthemes)

load("fullDivision.Rda")
load("data3N.Rda")
load("data2N.Rda")
load("USAirSum32N.Rda")
load("stateDelay.Rda")

data(flights)
data(weather)
```

```
ui <- navbarPage("Flights Analysis", inverse = TRUE,
                 tabPanel("Table Summary",
                          sidebarLayout(
                            sidebarPanel(

                  tags$head(
                      tags$style(HTML("
                        body {
                        background-color: #663399;
                        color: #330033;
                        }
                        "))
                  ),


    selectInput("origin", "Choose a origin airport:",
            choices = sort(unique(data2$OriginAirport)),
            selected = "John F. Kennedy International Airport"),

    selectInput("destination", "Choose a destination airport:",
            choices = sort(unique(data2$DestAirport)),
            selected = "San Francisco International Airport"),

    actionButton("go1", "Airport Flights Data",
  style="color: #ffffff;background-color: #663399;margin: 4px;"),

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

    selectInput("destState", "Choose a destination state:",
            choices = sort(unique(data2$DestFState)),
            selected = "California"),

    actionButton("go2", "State Flights Data",
 style="color: #ffffff;background-color: #663399;margin: 4px;"),

        HTML('<br/>', '<br/>'),

  strong("Table shows the mean departure delay in minutes
  from 2008-2016 for all flights with departure delay greater
  than 90 minutes with the specified origin and destination airport.
  Alternatively, origin and destination states can also be
  specified."), width = 3
```

```
                              ),
                            mainPanel(
                              dataTableOutput("view"),
                              verbatimTextOutput("avgDelayState"),
                              dataTableOutput("view2")
                            )
                            )
                  ),

                  tabPanel("Paths Analysis",
                           sidebarLayout(
                             sidebarPanel(
                               selectInput("responseP", "Choose a
                             response predictor:",
                               choices = as.character(c(2008,
            2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016))),

              selectInput("responseP2", "Choose an Origin state:",
        choices = sort(as.character(unique(group12$OriginFState))),
        "New York"),
              selectInput("responseP3", "Choose a Destination state:",
        choices = sort(as.character(unique(group12$DestFState))),
        "California"),
        actionButton("go111", "Departure Delays Mapping", style="color:
                     #ffffff;background-color: #663399;margin: 4px;"),
          br(),
      strong("Map shows the average departure delay in minutes from
2008-2016 for all United States flights with departure delay greater
than 90 minutes. Click a point to display airport level departure delay."),
width = 3
              ),
        mainPanel(
          leafletOutput("map4"),
          dataTableOutput("lat")
            )
            )
            ),
  navbarMenu("Delay by State, Division and Region from 2008-2016",
      tabPanel("State",
        sidebarLayout(
          sidebarPanel(
          selectInput("stateDest", "Choose a state:",
            choices = sort(as.character(unique(group12$OriginFState))),
            "New York"),
```

```
                br(),
          strong("Top plot shows departure delay in minutes for all
airports in the specified state from 2008-2016 for all flights with
departure delay greater than 90 minutes. Bottom plot displays the
aggregate departure delay for the specified state from 2008-2016."),
width = 2
          ),
                mainPanel(
                  plotlyOutput('plot31'),
                  plotlyOutput('plot32')
                  )
                  ),
              tabPanel("Midwest",
                sidebarLayout(
                  sidebarPanel(
                    strong("Top pair of plots shows average
departure delay in minutes for Midwestern United States over 2008-2016.
Bottom pair of plots shows aggregated departure delay from 2008 to 2016.
Left graph in both pairs corresponds to East North Central while right
graph corresponds to West North Central."), width = 0.5
                      ),
                      mainPanel(
                        plotlyOutput('plot34'),
                        plotlyOutput('plot35'), width = 12.5
                        )
                    )
                  ),
                  tabPanel("Northeast",
                           sidebarLayout(
                             sidebarPanel(
                    strong("Top pair of plots shows average
departure delay in minutes for Northeastern United States
over 2008-2016. Bottom pair of plots shows aggregated
departure delay from 2008 to 2016. Left graph in both
pairs corresponds to Middle Atlantic while right
graph corresponds to New England."), width = 0.5
                               ),
                             mainPanel(
                               plotlyOutput('plot36'),
                               plotlyOutput('plot37'), width = 12.5
                             )
                               )
                           ),
```

```
                tabPanel("South",
                  sidebarLayout(
                    sidebarPanel(
                      strong("Top three plots shows average
departure delay in minutes for Southern United States over
2008-2016. Bottom three plots show aggregated departure delay
from 2008 to 2016. Left graph in both trios corresponds to
East South Central, middle graph corresponds to
West South Central and right graph to South Atlantic."),
width = 0.5
          ),
          mainPanel(
              plotlyOutput('plot38'),
              plotlyOutput('plot39'), width = 12.5
                    )
                  )
                ),
                tabPanel("West",
                  sidebarLayout(
                    sidebarPanel(
                      strong("Top pair of plots shows average
departure delay in minutes for Western United States over 2008-2016.
Bottom pair of plots shows aggregated departure delay from 2008
to 2016. Left graph in both pairs corresponds to Pacific region while
right graph corresponds to Mountain region."), width = 0.5
                            ),
                          mainPanel(
                            plotlyOutput('plot40'),
                            plotlyOutput('plot41'), width = 12.5
                          )
                            )
                        )
                ),
navbarMenu("Delay by State and Region by Time and Carrier",
tabPanel("Delay by Hour",
  sidebarLayout(
    sidebarPanel(
      selectInput("stateDest21", "Choose a state:",
        choices = sort(as.character(unique(group18$OriginFState))),
        "New York"),
        strong("Specify state to observe departure delay by hour and
    region and aggregated departure delay for specified state from 2008
    to 2016."), width = 2
                    ),
```

```
                mainPanel(
                  plotlyOutput('plot61'),
                  plotlyOutput('plot42'), width = 10
                        )
                        )
                    ),
                tabPanel("Day of Week",
                    sidebarLayout(
                      sidebarPanel(
                      selectInput("stateDest22", "Choose a state:",
choices = sort(as.character(unique(group18$OriginFState))),
"New York"),
        strong("Departure delay by week."), width = 2
                ),
                mainPanel(
                    plotlyOutput('plot43'),
                    plotlyOutput('plot44')
                          )
                          )
                        ),
          tabPanel("Weekend Status",
                    sidebarLayout(
                      sidebarPanel(
                    selectInput("stateDest23", "Choose a state:",
choices = sort(as.character(unique(group18$OriginFState))),
"New York"),
    strong("Departure delay by weekend status."), width = 2
                ),
                mainPanel(
                    plotlyOutput('plot45'),
                    plotlyOutput('plot46')
                        )
                      )
                    ),
          tabPanel("Month",
                    sidebarLayout(
                      sidebarPanel(
                      selectInput("stateDest24", "Choose a state:",
choices = sort(as.character(unique(group18$OriginFState))),
"New York"),
    strong("Departure delay by month."), width = 2
            ),
                mainPanel(
                    plotlyOutput('plot47'),
```

```
                  plotlyOutput('plot48')
                    )
                  )
                ),
              tabPanel("Season",
                  sidebarLayout(
                    sidebarPanel(
                    selectInput("stateDest25", "Choose a state:",
    choices = sort(as.character(unique(group18$OriginFState))),
    "New York"),
        strong("Departure delay by season."), width = 2
          ),
              mainPanel(
                 plotlyOutput('plot49'),
                 plotlyOutput('plot50')
                    )
                  )
                ),
              tabPanel("Carrier",
                  sidebarLayout(
                    sidebarPanel(
                    selectInput("stateDest26", "Choose a state:",
    choices = sort(as.character(unique(group18$OriginFState))),
    "New York"),
        strong("Departure delay by carrier."), width = 0.5
          ),
               mainPanel(
                  plotlyOutput('plot51'),
                  plotlyOutput('plot52'), width = 12
                    )
                  )
                )
              ),
    tabPanel("Aggregate Delay by Region",
        sidebarLayout(
          sidebarPanel(
            selectInput("responseM", "Choose a response predictor:",
        choices = as.character(c(2008, 2009, 2010, 2011, 2012,
                                 2013, 2014, 2015, 2016))),
        selectInput("responseM2", "Choose a US region:",
          choices = sort(as.character(c("East North Central",
                                        "East South Central",
                          "Middle Atlantic", "Mountain",
                          "New England", "Pacific",
```

```
                          "South Atlantic",
                          "West North Central",
                          "West South Central")))),
        br(),
        strong("Map shows the average departure delay greater
than 90 minutes for United States airports from 2008 to 2016.
Hover over a point to display airport level departure delay.")
               ),
               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_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$yearEnd) %>%
      mutate(yearF = as.factor(year)) %>%
      dplyr::group_by_("yearF", input$response) %>%
      dplyr::summarise(MeanDep = mean(dep_delay))
    return(DatCarrier)
  })


  output$plot <- renderPlotly({
    withProgress(message = "Application loading", value = 0, {
```

```
    dfweek <-  df_weekend()
    incProgress(0.6, detail = "Building plot")
    p = ggplot(dfweek, aes_string(x = "yearF", y = "MeanDep",
                                  group = input$response,
                                  color = input$response)) +
      geom_point() + geom_line(size = 1) +
      ggtitle(paste("Mean departure delay overtime by",
                    input$response))
    incProgress(0.4, detail = "Finishing...")
    ggplotly(p)
  })
})

output$plot3 <- renderPlotly({
  USAirSum32 <- subset(USAirSum32, year == input$responseM &
                         Division == input$responseM2)
  USAirSum33 <- USAirSum32
  USAirSum33 <- plyr::rename(USAirSum33, replace =
                               c("mean2" = "AverageDelay"))
  g <- list(
    scope = 'usa',
    projection = list(type = 'albers usa'),
    showland = TRUE,
    landcolor = toRGB("gray85"),
    subunitwidth = 1,
    countrywidth = 1,
    subunitcolor = toRGB("white"),
    countrycolor = toRGB("white")
  )
  USAirSum33$AverageDelay <- round(USAirSum33$AverageDelay, 2)
  p <- plot_geo(USAirSum33, locationmode = 'USA-states',
                sizes = c(1, 250)) %>%
    add_markers(x = ~ OriginLong, y = ~ OriginLat,
                color = ~ AverageDelay, alpha= 0.8,
                text = ~ paste(OriginFState, "<br />",
                               AverageDelay, "<br />",
                               OriginAirport)) %>%
    layout(title = (paste('Mean Departure Delay by Airport for',
                          unique(USAirSum33$year), 'and',
                          unique(USAirSum33$Division))),
           geo = g)
  #plot_ly(p)
  ggplotly(p)
  #ggplotly(p)
```

```
})


datMap2 <- eventReactive(input$go111, {
  group13 <- subset(group12, year == input$responseP &
                      OriginFState ==
                      input$responseP2 & DestFState ==
                      input$responseP3)
  group13$OriginLong <- round(group13$OriginLong, 4)
  group13$OriginLat <- round(group13$OriginLat, 4)

  group13$DestLong <- round(group13$DestLong, 4)
  group13$DestLat <- round(group13$DestLat, 4)
  return(group13)
})

output$map4 <- renderLeaflet({
  withProgress(message = "Application loading", value = 0, {
    group13 <- datMap2()
    map4 <- leaflet() %>% addTiles()
    incProgress(0.7, detail = "Building plot")
    if (nrow(group13) != 0)
    {
      for (i in 1:nrow(group13))
      {
        long <- cbind(group13[i,"OriginLong"],
                      group13[i,"DestLong"])
        lat <- cbind(group13[i,"OriginLat"],
                     group13[i,"DestLat"])
        long1 <- as.list(data.frame(t(long)))
        lat1 <- as.list(data.frame(t(lat)))

        map4 <- map4 %>% addTiles(options =
                  providerTileOptions(noWrap = TRUE)) %>%
          addCircleMarkers(lng = long1$t.long.,
        lat = lat1$t.lat., group='circles', color = "#660099",
        fillColor = "black",
        weight = (group13[i,"meanDelay"])/20) %>%
          addPolylines(lng = long1$t.long., lat = lat1$t.lat.,
                color = "#660099")
      }
      return(map4)
    }
    else
```

```
    {
      return(map4)
    }
    incProgress(0.3, detail = "Finishing...")
  })
})

observeEvent(input$map4_marker_click,{
  group13 <- datMap2()
  p <- input$map4_marker_click
  clat <- p$lat
  clng <- p$lng
  clat <- round(clat, 4)
  clng <- round(clng, 4)
  group14 <- subset(group13,
                    (OriginLong == clng & OriginLat == clat) |
                      (DestLong == clng & DestLat == clat))
  output$latitude <- renderText({return(clat)})
  output$longitude <- renderText({return(clng)})
  group15 <- group14 %>%
    select(OriginAirport, OriginFState, DestAirport,
                     DestFState, meanDelay)
  group15 <- plyr::rename(group15, replace =
                            c("OriginFState" = "OriginState"))
  group15 <- plyr::rename(group15, replace =
                            c("DestFState" = "DestState"))
  output$lat <- renderDataTable({return(group15)})
})

#Midwest plot
output$plot34 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    M <- subset(group12, OrigRegion == "Midwest")
    MGroup <- M %>% group_by(OriginFState, year, OrigDivision) %>%
      summarise(AvgDelay = mean(meanDelay))

    MGroup <- plyr::rename(MGroup, replace =
                             c("OriginFState" = "State"))

    MidAt <- subset(MGroup, OrigDivision == "East North Central")
    NewEng <- subset(MGroup, OrigDivision == "West North Central")

    incProgress(0.6, detail = "Building plot")
```

```
    gra <- ggplot(data = MidAt, aes(x=year, y=AvgDelay))  +
      geom_line(aes(colour = State), size = 1) +
      theme(legend.position="right") +
      theme(legend.title=element_blank()) +
      theme_fivethirtyeight() + scale_colour_few()
    g <- ggplotly(gra)


    gra1 <- ggplot(data = NewEng, aes(x=year, y=AvgDelay)) +
      geom_line(aes(colour = State), size = 1) +
      theme(legend.position="right") + ylab("Year") +
      theme(legend.title=element_blank()) +
      theme_fivethirtyeight() + scale_colour_few() +
      xlab("AvgDelay")
    g1 <- ggplotly(gra1)

    p <- subplot(g, g1)
    incProgress(0.3, detail = "Finishing...")
    p
  })
})


output$plot35 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    M <- subset(group12, OrigRegion == "Midwest")
    MGroup <- M %>% group_by(OriginFState, year, OrigDivision) %>%
      summarise(AvgDelay = mean(meanDelay))

    MidAt <- subset(MGroup, OrigDivision == "East North Central")
    NewEng <- subset(MGroup, OrigDivision == "West North Central")

    incProgress(0.7, detail = "Building plot")

    MidAt2 <- MidAt %>% group_by(OriginFState) %>%
      summarise(AvgDelay = mean(AvgDelay))

    NewEng2 <- NewEng %>% group_by(OriginFState) %>%
      summarise(AvgDelay = mean(AvgDelay))

    incProgress(0.3, detail = "Finishing...")
    g2 <- ggplot(MidAt2, aes(x = OriginFState, y = AvgDelay)) +
      geom_bar(fill="#330033", stat = 'identity', width = 0.4) +
      theme_fivethirtyeight() + scale_colour_few()
```

```
      g2 <- ggplotly(g2)
      g2

      g3 <- ggplot(NewEng2, aes(x= OriginFState, y = AvgDelay)) +
        geom_bar(fill="#330033", stat = 'identity', width = 0.5) +
        theme_fivethirtyeight() + scale_colour_few()
      g3 <- ggplotly(g3)
      g3

      incProgress(0.6, detail = "Finishing...")
      p2 <- subplot(g2, g3)
      p2
    })
  })

output$plot36 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    NE <- subset(group12, OrigRegion == "Northeast")
    NEGroup <- NE %>% group_by(OriginFState, year, OrigDivision) %>%
      summarise(AvgDelay = mean(meanDelay))

    MidAt <- subset(NEGroup, OrigDivision == "Middle Atlantic")
    NewEng <- subset(NEGroup, OrigDivision == "New England")

    MidAt <- plyr::rename(MidAt, replace = c("OriginFState" = "State"))
    gra <- ggplot(data = MidAt, aes(x=year, y=AvgDelay)) +
      geom_line(aes(colour = State), size = 1) +
      theme(legend.position="right") +
      theme(legend.title=element_blank()) + theme_fivethirtyeight() +
      scale_colour_few()

    incProgress(0.7, detail = "Building plot")

    g <- ggplotly(gra)
    g

    NewEng <- plyr::rename(NewEng, replace = c("OriginFState" = "State"))
    gra1 <- ggplot(data = NewEng, aes(x=year, y=AvgDelay)) +
      geom_line(aes(colour = State), size = 1) +
      theme(legend.position="right") +
      theme(legend.title=element_blank()) + theme_fivethirtyeight() +
      scale_colour_few()

    g1 <- ggplotly(gra1)
```

```
    incProgress(0.3, detail = "Finishing...")
    subplot(g, g1)
  })
  })

output$plot37 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    NE <- subset(group12, OrigRegion == "Northeast")
    NEGroup <- NE %>% group_by(OriginFState, year, OrigDivision) %>%
      summarise(AvgDelay = mean(meanDelay))

    MidAt <- subset(NEGroup, OrigDivision == "Middle Atlantic")
    NewEng <- subset(NEGroup, OrigDivision == "New England")

    MidAt2 <- MidAt %>% group_by(OriginFState) %>%
      summarise(AvgDelay = mean(AvgDelay))

    NewEng2 <- NewEng %>% group_by(OriginFState) %>%
      summarise(AvgDelay = mean(AvgDelay))

    incProgress(0.7, detail = "Building plot")

    g2 <- ggplot(MidAt2, aes(OriginFState)) +
      geom_bar(fill="#330033", aes(weight = AvgDelay), width = 0.4) +
      theme_fivethirtyeight() + scale_colour_few()

    g2 <- ggplotly(g2)

    g3 <- ggplot(NewEng2, aes(OriginFState)) +
      geom_bar(fill="#330033", aes(weight = AvgDelay), width = 0.4) +
      theme_fivethirtyeight() + scale_colour_few()
    g3 <- ggplotly(g3)

    incProgress(0.3, detail = "Finishing...")
    subplot(g2, g3)
  })
})

output$plot38 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    NE <- subset(group12, OrigRegion == "South")
    NEGroup <- NE %>% group_by(OriginFState, year, OrigDivision) %>%
      summarise(AvgDelay = mean(meanDelay))
```

```
    NEGroup <- plyr::rename(NEGroup, replace =
                              c("OriginFState" = "State"))

    MidAt <- subset(NEGroup, OrigDivision == "East South Central")
    NewEng <- subset(NEGroup, OrigDivision == "West South Central")
    NewSouth <- subset(NEGroup, OrigDivision == "South Atlantic")

    incProgress(0.7, detail = "Building plot")

    gra <- ggplot(data = MidAt, aes(x=year, y=AvgDelay)) +
      geom_line(aes(colour = State), size = 1) +
      theme(legend.position="right") +
      theme(legend.title=element_blank()) +
      theme_fivethirtyeight() + scale_colour_hue()

    g <- ggplotly(gra)

    gra1 <- ggplot(data = NewEng, aes(x=year, y=AvgDelay)) +
      geom_line(aes(colour = State), size = 1) +
      theme(legend.position="right") +
      theme(legend.title=element_blank()) +
      theme_fivethirtyeight() + scale_colour_hue()

    g1 <- ggplotly(gra1)

    gra2 <- ggplot(data = NewSouth, aes(x=year, y=AvgDelay)) +
      geom_line(aes(colour = State), size = 1) +
      theme(legend.position="right") +
      theme(legend.title=element_blank()) +
      theme_fivethirtyeight() + scale_colour_hue()
    g2 <- ggplotly(gra2)

    incProgress(0.3, detail = "Finishing...")
    subplot(g, g1, g2)
  })
})

output$plot39 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    NE <- subset(group12, OrigRegion == "South")
    NEGroup <- NE %>% group_by(OriginFState, year, OrigDivision) %>%
      summarise(AvgDelay = mean(meanDelay))

    MidAt <- subset(NEGroup, OrigDivision == "East South Central")
```

```
    NewEng <- subset(NEGroup, OrigDivision == "West South Central")
    NewSouth <- subset(NEGroup, OrigDivision == "South Atlantic")

    MidAt2 <- MidAt %>% group_by(OriginFState) %>%
      summarise(AvgDelay = mean(AvgDelay))

    NewEng2 <- NewEng %>% group_by(OriginFState) %>%
      summarise(AvgDelay = mean(AvgDelay))

    NewSouth2 <- NewSouth %>% group_by(OriginFState) %>%
      summarise(AvgDelay = mean(AvgDelay))

    incProgress(0.7, detail = "Building plot")

    g2 <- ggplot(MidAt2, aes(OriginFState)) +
      geom_bar(fill="#330033", aes(weight = AvgDelay), width = 0.4) +
      theme_fivethirtyeight() + scale_colour_few()
    g2 <- ggplotly(g2)

    g3 <- ggplot(NewEng2, aes(OriginFState)) +
      geom_bar(fill="#330033", aes(weight = AvgDelay), width = 0.4) +
      theme_fivethirtyeight() + scale_colour_few()
    g3 <- ggplotly(g3)

    g4 <- ggplot(NewSouth2, aes(OriginFState)) +
      geom_bar(fill="#330033", aes(weight = AvgDelay), width = 0.4) +
      theme_fivethirtyeight() + scale_colour_few()
    g4 <- ggplotly(g4)

    incProgress(0.3, detail = "Finishing...")

    subplot(g2, g3, g4)
  })
})

output$plot40 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    M <- subset(group12, OrigRegion == "West")
    MGroup <- M %>% group_by(OriginFState, year, OrigDivision) %>%
      summarise(AvgDelay = mean(meanDelay))

    MGroup <- plyr::rename(MGroup, replace = c("OriginFState" = "State"))

    MidAt <- subset(MGroup, OrigDivision == "Pacific")
```

```
    NewEng <- subset(MGroup, OrigDivision == "Mountain")

    incProgress(0.6, detail = "Building plot")

    gra <- ggplot(data = MidAt, aes(x=year, y=AvgDelay)) +
      geom_line(aes(colour = State), size = 1) +
      theme(legend.position="right") +  theme(legend.title=element_blank()) +
      theme_fivethirtyeight() + scale_colour_hue()
    g <- ggplotly(gra)

    gra1 <- ggplot(data = NewEng, aes(x=year, y=AvgDelay)) +
      geom_line(aes(colour = State), size = 1) +
      theme(legend.position="right") +  theme(legend.title=element_blank()) +
      theme_fivethirtyeight() + scale_colour_hue()
    g1 <- ggplotly(gra1)

    incProgress(0.3, detail = "Finishing...")
    subplot(g, g1)
  })
})

output$plot41 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    M <- subset(group12, OrigRegion == "West")
    MGroup <- M %>% group_by(OriginFState, year, OrigDivision) %>%
      summarise(AvgDelay = mean(meanDelay))

    MidAt <- subset(MGroup, OrigDivision == "Pacific")
    NewEng <- subset(MGroup, OrigDivision == "Mountain")

    MidAt2 <- MidAt %>% group_by(OriginFState) %>%
      summarise(AvgDelay = mean(AvgDelay))

    NewEng2 <- NewEng %>% group_by(OriginFState) %>%
      summarise(AvgDelay = mean(AvgDelay))

    incProgress(0.6, detail = "Building plot")

    g2 <- ggplot(MidAt2, aes(OriginFState)) +
      geom_bar(fill="#330033", aes(weight = AvgDelay), width = 0.4) +
      theme_fivethirtyeight() + scale_colour_few()
    g2 <- ggplotly(g2)

    g3 <- ggplot(NewEng2, aes(OriginFState)) +
```

```
      geom_bar(fill="#330033", aes(weight = AvgDelay), width = 0.4) +
      theme_fivethirtyeight() + scale_colour_few()
    g3 <- ggplotly(g3)

    incProgress(0.3, detail = "Finishing...")
    subplot(g2, g3)
  })
})

  output$plot31 <- renderPlotly({
    withProgress(message = "Application loading", value = 0, {
      group13 <- subset(group12, OriginFState == input$stateDest)
      group14 <- group13 %>% group_by(OriginAirport, year, OriginFState) %>%
        summarise(AvgDelay = mean(meanDelay))
      group14$OriginAirport <- as.factor(group14$OriginAirport)
      incProgress(0.6, detail = "Building plot")
      plot <- ggplot(data = group14, aes(x=year, y=AvgDelay)) +
        geom_line(size = 1)  +
        aes(colour=OriginAirport) + theme(legend.position="none") +
        theme(legend.position="right") + theme_fivethirtyeight() +
        scale_colour_hue()
      incProgress(0.4, detail = "Finishing...")
      ggplotly(plot)
    })
  })


  output$plot32 <- renderPlotly({
    withProgress(message = "Application loading", value = 0, {
      group13 <- subset(group12, OriginFState == input$stateDest)
      group14 <- group13 %>% group_by(OriginAirport, year, OriginFState) %>%
        summarise(AvgDelay = mean(meanDelay))
      group14$OriginAirport <- as.factor(group14$OriginAirport)
      incProgress(0.6, detail = "Building plot")
      group15 <- group14 %>% group_by(year, OriginFState) %>%
        summarise(AvgDelay = mean(AvgDelay))
      graph <- ggplot(data = group15, aes(x=year, y=AvgDelay)) +
        geom_line(color = "#330033", size = 1) +
        labs(title=paste("Departure Delay for", input$stateDest, sep = " ")) +
        theme_fivethirtyeight() + scale_colour_few()
      incProgress(0.6, detail = "Finishing...")
      ggplotly(graph)
    })
  })
```

```
output$view <- renderDataTable(df_subset())

output$avgDelayState <- renderText({
  dataDel <- df_subset2()
  meanDepDelay <- mean(dataDel$meanDelay)
  text <- paste("Average Departure Delay from ", dataDel$OriginFState[1]," to ",
                dataDel$DestFState[1], ": ", round(meanDepDelay, 3), sep = "")
  return(text)
})

output$view2 <- renderDataTable(df_subset2())

output$plot61 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    incProgress(0.6, detail = "Building plot")
    group19 <- group18 %>% group_by(hour, OrigRegion) %>%
      summarise(AvgDelay = mean(dep_delay))
    r1 <- ggplot(data = group19, aes(x=hour, y=AvgDelay,
                                     group = OrigRegion)) +
      geom_line(aes(colour = OrigRegion), size = 1) +
      theme_fivethirtyeight() + scale_colour_few()
    incProgress(0.3, detail = "Finishing...")
    ggplotly(r1)
  })
})

output$plot42 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    group20 <- subset(group18, OriginFState == input$stateDest21)
    group20 <- subset(group18, OriginFState == "New York")
    group19 <- group20 %>% group_by(hour) %>% summarise(AvgDelay =
                                                          mean(dep_delay))
    incProgress(0.6, detail = "Building plot")
    r1 <- ggplot(data = group19, aes(x=hour, y=AvgDelay, group = 1)) +
      geom_line(size = 1, color = "#330033") +
      labs(title=paste("Departure Delay for", input$stateDest21, sep = " ")) +
      theme_fivethirtyeight() + scale_colour_few()
    incProgress(0.4, detail = "Finishing...")
    ggplotly(r1)
  })
})

output$plot43 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
```

```
    group18 <- plyr::rename(group18, replace = c("OrigRegion" = "Region"))
    group19 <- group18 %>% group_by(week, Region) %>%
      summarise(AvgDelay = mean(dep_delay))
    incProgress(0.6, detail = "Building plot")
    r1 <- ggplot(data = group19, aes(x = week,
                                     y = AvgDelay, group = Region)) +
      geom_line(aes(colour = Region), size = 1) +
      theme_fivethirtyeight() + scale_colour_few()
    incProgress(0.4, detail = "Finishing...")
    ggplotly(r1)
  })
})

output$plot44 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    group20 <- subset(group18, OriginFState == input$stateDest22)
    group19 <- group20 %>% group_by(week) %>%
      summarise(AvgDelay = mean(dep_delay))
    incProgress(0.6, detail = "Building plot")
    r1 <- ggplot(data = group19, aes(x=week, y=AvgDelay, group = 1)) +
      geom_line(size = 1, color = "#330033") +
      labs(title=paste("Departure Delay for",
                       input$stateDest22, sep = " ")) +
      theme_fivethirtyeight() + scale_colour_few()
    incProgress(0.4, detail = "Finishing...")
    ggplotly(r1)
  })
})

output$plot45 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    group18 <- plyr::rename(group18, replace = c("OrigRegion" = "Region"))
    group19 <- group18 %>% group_by(weekend, Region) %>%
      summarise(AvgDelay = mean(dep_delay))
    incProgress(0.7, detail = "Building plot")
    r1 <- ggplot(data = group19, aes(x=weekend,
                                     y=AvgDelay, group = Region)) +
      geom_line(aes(colour = Region), size = 1) +
      theme_fivethirtyeight() + scale_colour_few()
    incProgress(0.3, detail = "Finishing...")
    ggplotly(r1)
  })
})
```

```
output$plot46 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    group20 <- subset(group18, OriginFState == input$stateDest23)
    group19 <- group20 %>% group_by(weekend) %>%
      summarise(AvgDelay = mean(dep_delay))
    incProgress(0.6, detail = "Building plot")
    r1 <- ggplot(data = group19, aes(x=weekend, y=AvgDelay, group = 1)) +
      geom_line(size = 1, color = "#330033") +
      labs(title=paste("Departure Delay for",
                       input$stateDest23, sep = " ")) +
      theme_fivethirtyeight() + scale_colour_few()
    incProgress(0.6, detail = "Finishing...")
    ggplotly(r1)
  })
})

output$plot47 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    group18 <- plyr::rename(group18, replace = c("OrigRegion" = "Region"))
    group19 <- group18 %>% group_by(month, Region) %>%
      summarise(AvgDelay = mean(dep_delay))
    incProgress(0.6, detail = "Building plot")
    r1 <- ggplot(data = group19, aes(x=month,
                                     y=AvgDelay, group = Region)) +
      geom_line(aes(colour = Region), size = 1) +
      theme_fivethirtyeight() + scale_colour_few()
    incProgress(0.4, detail = "Finishing...")
    ggplotly(r1)
  })
})

output$plot48 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    group20 <- subset(group18, OriginFState == input$stateDest24)
    group19 <- group20 %>% group_by(month) %>%
      summarise(AvgDelay = mean(dep_delay))
    incProgress(0.6, detail = "Building plot")
    r1 <- ggplot(data = group19, aes(x=month, y=AvgDelay, group = 1)) +
      geom_line(size = 1, color = "#330033") +
      labs(title=paste("Departure Delay for",
                       input$stateDest24, sep = " ")) +
      theme_fivethirtyeight() + scale_colour_few()
    incProgress(0.4, detail = "Finishing...")
    ggplotly(r1)
```

```
  })
})

output$plot49 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    group18 <- plyr::rename(group18, replace = c("OrigRegion" = "Region"))
    group19 <- group18 %>% group_by(season, Region) %>%
      summarise(AvgDelay = mean(dep_delay))
    incProgress(0.6, detail = "Building plot")
    r1 <- ggplot(data = group19, aes(x=season,
                                     y=AvgDelay, group = Region)) +
      geom_line(aes(colour = Region), size = 1) +
      theme_fivethirtyeight() + scale_colour_few()
    incProgress(0.4, detail = "Finishing...")
    ggplotly(r1)
  })
})

output$plot50 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    group20 <- subset(group18, OriginFState == input$stateDest25)
    group19 <- group20 %>% group_by(season) %>%
      summarise(AvgDelay = mean(dep_delay))
    incProgress(0.6, detail = "Building plot")
    r1 <- ggplot(data = group19, aes(x=season,
                                     y=AvgDelay, group = 1)) +
      geom_line(size = 1, color = "#330033") +
      labs(title=paste("Departure Delay for",
                       input$stateDest25, sep = " ")) +
      theme_fivethirtyeight() + scale_colour_few()
    incProgress(0.4, detail = "Finishing...")
    ggplotly(r1)
  })
})

output$plot51 <- renderPlotly({
  withProgress(message = "Application loading", value = 0, {
    group18 <- plyr::rename(group18, replace = c("OrigRegion" = "Region"))
    group19 <- group18 %>% group_by(carrier, Region) %>%
      summarise(AvgDelay = mean(dep_delay))
    incProgress(0.6, detail = "Building plot")
    r1 <- ggplot(data = group19, aes(x=carrier,
                                     y=AvgDelay, group = Region)) +
      geom_line(aes(colour = Region), size = 1) +
```

```
        theme_fivethirtyeight() + scale_colour_few()
      incProgress(0.4, detail = "Finishing...")
      ggplotly(r1)
    })
  })

  output$plot52 <- renderPlotly({
    withProgress(message = "Application loading", value = 0, {
      group20 <- subset(group18, OriginFState == input$stateDest26)
      group19 <- group20 %>% group_by(carrier) %>%
        summarise(AvgDelay = mean(dep_delay))
      incProgress(0.6, detail = "Building plot")
      r1 <- ggplot(data = group19, aes(x=carrier,
                                       y=AvgDelay, group = 1)) +
        geom_line(size = 1, color = "#330033") +
        labs(title=paste("Departure Delay for",
                         input$stateDest26, sep = " ")) +
        theme_fivethirtyeight() + scale_colour_few()
      incProgress(0.4, detail = "Finishing...")
      ggplotly(r1)
    })
  })

})

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

Web Scraping for Airport and State Names

```
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)
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)
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

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (n.d.). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems 5 (NIPS 2012). Retrieved from https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

LeCun, Y., Bengio, Y., & Hinton, G. (2015, May). Deep learning. Nature. Retrieved from http://www.nature.com/nature/journal/v521/n7553/abs/nature14539.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