# Major assignment 2: Your submission

This is your assignment template for [AnalyticsX Major assignment 2](https://courses.edx.org/courses/course-v1:AdelaideX+AnalyticsX+3T2018/courseware/e0b375053b6441a08d658133752b3531/5965d914c42540e58416a76fb1f502ba/1?activate_block_id=block-v1%3AAdelaideX%2BAnalyticsX%2B3T2018%2Btype%40vertical%2Bblock%4000cc761eea174adbb4be63d3ec1c7484). Save this document on our local machine and include all of your work within the relevant sections. Once you’ve completed all five parts of the assignment, upload the document via the submission area on the “[Submit your assignment](https://courses.edx.org/courses/course-v1:AdelaideX+AnalyticsX+3T2018/courseware/e0b375053b6441a08d658133752b3531/5965d914c42540e58416a76fb1f502ba/6?activate_block_id=block-v1%3AAdelaideX%2BAnalyticsX%2B3T2018%2Btype%40vertical%2Bblock%40cb1643bf82444588a962216431740cd7)” page at the end of Major assignment 1.

# Checklist

* Have you shown all of your working?
* Have you given all numbers to the correct number of decimal places?
* Have you included all R output and plots to support your answers where necessary?
* Have you included all of your R code – input and output?
* Have you made sure that all plots and tables each have a meaningful caption?

**Quick links:**

[Major assignment 2: Part 1](#_Major_assignment_2:)

[Major assignment 2: Part 2](#_Major_assignment_2:_1)

[Major assignment 2: Part 3](#_Major_assignment_2:_2)

[Major assignment 2: Part 4](#_Major_assignment_2:_3)

# Major assignment 2: Part 1

1. Load in the flights dataset as a Spark table [4 points]

Your input code and output code go here:

**Load the required libraries and create the Spark connection**

library(dplyr)

library(sparklyr)

library(ggplot2)

library(nycflights13)

sc <- spark\_connect(master = "local")

**Calculate the mean airtime for each carrier**

flights\_tbl <- copy\_to(sc, nycflights13::flights, "flights", overwrite=TRUE)

mean\_airtime <- flights\_tbl %>%

  group\_by(carrier) %>%

  summarise(mean\_airtime = mean(air\_time)) %>%

  collect

print.data.frame(mean\_airtime)

Table 1. Mean Airtime by Carrier.

|  |
| --- |
| carrier mean\_airtime  1 EV 90.07619  2 US 88.57380  3 WN 147.82481  4 VX 337.00235  5 YV 65.74081  6 UA 211.79135  7 DL 173.68880  8 MQ 91.18025  9 OO 83.48276  10 B6 151.17717  11 F9 229.59912  12 AA 188.82230  13 FL 101.14394  14 AS 325.61777  15 9E 86.78160  16 HA 623.08772 |

1. Create a new categorical variable long\_flight which is 1 when a flight is further than 4,000 miles. [2 points]

Your input code and output code go here:

long\_flight\_feature <- flights\_tbl %>%

  ft\_binarizer("distance", "long\_flight", threshold = 4000)

long\_flight\_feature %>%

  filter(long\_flight == 1) %>%

  summarise(count=n())

Table 2. Number of flights further than 4000 miles.

|  |
| --- |
| # Source: spark<?> [?? x 1]  count  <dbl>  1 707 |

1. Perform a principle component analysis using the variables flight time, distance, and departure time. [3 points]

Your input code and output code go here:

pca\_model <- tbl(sc, "flights") %>%

  select(air\_time, distance, dep\_time) %>%

  na.omit() %>%

  ml\_pca()

print(pca\_model)

Table 3. PCA Model.

|  |
| --- |
| \* Dropped 9430 rows with 'na.omit' (336776 => 327346)  Explained variance:  PC1 PC2 PC3  0.6975982637 0.3021978609 0.0002038754  Rotation:  PC1 PC2 PC3  air\_time -0.12514862 0.001940123 -0.9921361086  distance -0.99200200 0.016312867 0.1251636066  dep\_time 0.01642742 0.999865054 -0.0001169274 |

1. Use ggplot to visualise the results. [3 points]

Your input code and output code, including your plot, go here:

D <- as.matrix(flights[c("air\_time", "distance", "dep\_time")])

E <- as.matrix(pca\_model$pc)

P <-  D %\*% E

PCs <- as.data.frame(P)

ggplot(PCs,aes(PC1,PC2)) +

  geom\_point()

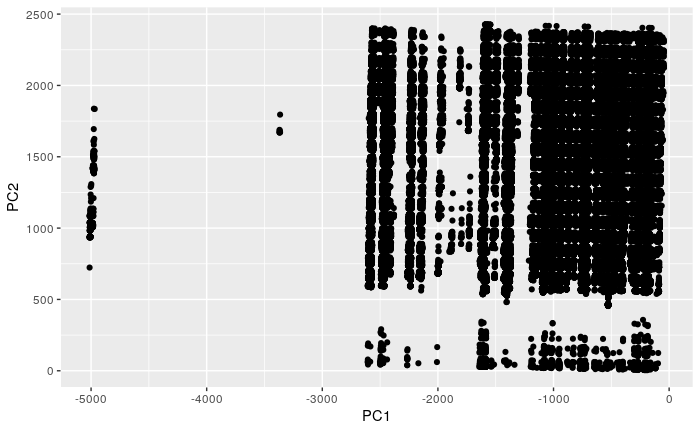


Figure 1. Scatterplot of Principle Component 1 vs 2.

1. Use your plot to create another plot to explain the observed clustering of the data. [2 points]

Your input code and output code, including your plot, go here:

**Data Preparation**

long\_flight\_feature\_data <- long\_flight\_feature %>%

  collect()

PCs$long\_flight <- long\_flight\_feature\_data$long\_flight

PCs$dep\_time <-long\_flight\_feature\_data$dep\_time

PCs$air\_time <-long\_flight\_feature\_data$air\_time

**Create Scatterplot**

ggplot(PCs,aes(PC1,PC2)) +

  geom\_point(aes(colour=long\_flight))

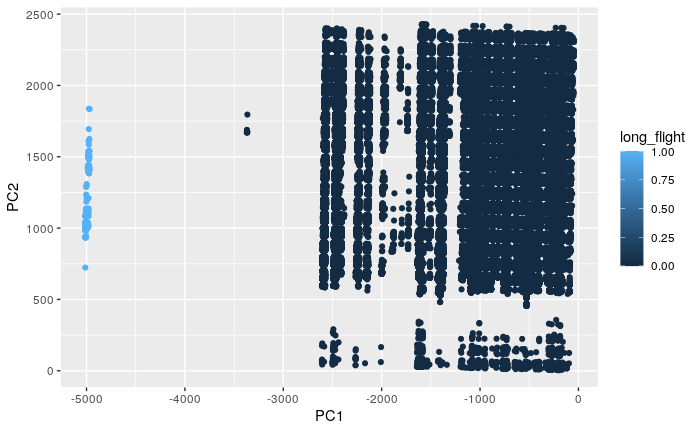
****

Figure 2. Scatterplot of PC1 vs PC2 using long\_flight as the colour to identify clusters.

# Major assignment 2: Part 2

1. Set the seed to be 42, then create a binary variable late\_arrival which is 1 if a flight arrives more than 30 minutes late. Count the number of late arrivals. [3 points]

Your input code and output code go here:

Connect to Spark

sc <- spark\_connect(master = "local")

Data Preparation

flights\_tbl <- copy\_to(sc, nycflights13::flights, "flights", overwrite=TRUE)

flights\_final\_tbl <- flights\_tbl %>%

  ft\_binarizer("arr\_delay", "late\_arrival", threshold=30)

Summarise by the late\_arrival feature

flights\_final\_tbl %>%

  group\_by(late\_arrival) %>%

  summarise(count=count())

Table 4. Feature Summary - 1 indicates the flight is more than 30 minutes late, NA indicates the total number of rows that did not have a value in the arr\_delay column.

|  |
| --- |
| # Source: spark<?> [?? x 2]  late\_arrival count  <dbl> <dbl>  1 0 275847  2 1 51499  3 NA 9430 |

1. Use logistic regression to predict late arrivals based on carrier, departure delay, month, and year. [4 points]

Your input code and output code go here:

Create Training and Testing partitions

partitions <- flights\_final\_tbl %>%

  na.omit() %>%

  sdf\_random\_split(training=0.75, test=0.25, seed=42)

train\_tbl <- partitions$train

test\_tbl <- partitions$test

Create the Logistic Regression Model

ml\_log <- test\_tbl %>%

  ml\_logistic\_regression(late\_arrival ~ carrier + dep\_delay + month + year)

summary(ml\_log)

Table 5. Model Coefficients.

|  |
| --- |
| Coefficients:  (Intercept) carrier\_UA carrier\_B6 carrier\_EV carrier\_DL carrier\_AA carrier\_MQ carrier\_US  -2.4139389394 -1.3433477489 -0.9729977259 -1.1547489938 -1.3101493616 -1.1989586582 -0.5830090254 -0.9090248218  carrier\_9E carrier\_WN carrier\_VX carrier\_FL carrier\_AS carrier\_F9 carrier\_YV carrier\_HA  -1.4260723295 -1.6457643720 -1.4976918640 -0.8197291012 -1.1819448199 -0.4597504879 -0.2876300466 -3.5570025754  dep\_delay month year  0.0985114898 0.0008124852 0.0000000000 |

1. Produce a ROC curve for your model. [4 points]

Your input and output code goes here:

Perform Predictions on the test dataset

pred\_lr <- ml\_predict(ml\_log, test\_tbl) %>% collect

pred\_lr$p1 <- unlist(pred\_lr$probability)[ c(FALSE,TRUE) ]

Calculate the values using the function given in Section 7 and plot the ROC curve

ROC\_lr <- get\_roc(L = pred\_lr$late\_arrival, f = pred\_lr$p1)

ggplot(ROC\_lr, aes(x = FPR, y = TPR)) + geom\_line(aes(col = "LR prediction"))

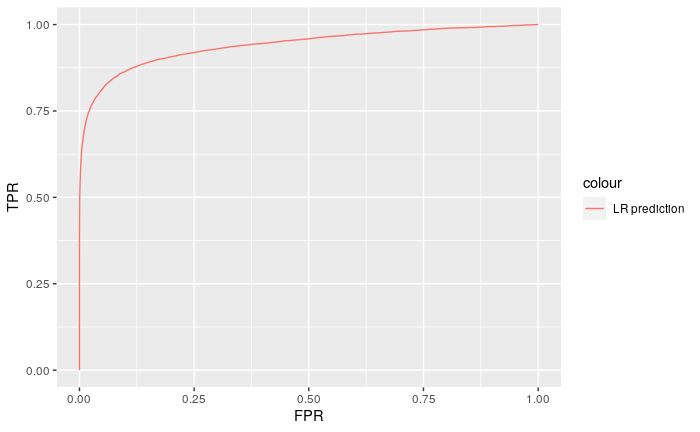


Figure 3. ROC curve of the Late Arrival model.

1. Remove each predictor variable individually to determine which is the best single predictor of late arrivals. [3 points]

Your input and output code go here:

I found this question a little confusing, but I also struggled with everything else up to this point that doesn’t quite seem right to me, so I might have misunderstood a number of things. If I were to do this outside of the course restrictions, I might have first tried to create a logistic regression for each predictor variable on its own and then simply look at the model evaluation of each for comparison. For example:

ml\_log <- test\_tbl %>%

  ml\_logistic\_regression(late\_arrival ~ dep\_delay)

pred <- ml\_predict(ml\_log, test\_tbl)

ml\_binary\_classification\_evaluator(pred, metric\_name = "areaUnderROC")

**Remove one predictor variable at a time and plot a ROC curve**

roc1 <- calculate\_roc(formula(late\_arrival ~ carrier + dep\_delay + month + year), test\_tbl)

roc2 <- calculate\_roc(formula(late\_arrival ~ dep\_delay + month + year), test\_tbl)

roc3 <- calculate\_roc(formula(late\_arrival ~ month + year), test\_tbl)

roc4 <- calculate\_roc(formula(late\_arrival ~ year), test\_tbl)

ggplot(roc1, aes(x=FPR, y=TPR)) +

  geom\_line(aes(col="carrier + dep\_delay + month + year")) +

  geom\_line(data=roc2, aes(col="dep\_delay + month + year")) +

  geom\_line(data=roc3, aes(col="month + year")) +

  geom\_line(data=roc4, aes(col="year"))

Some of the conclusions from the ROC plots below should come as no surprise.

* The dataset is for 2013 flight, so we can therefore already conclude that the year feature will have no predictive power in the model.
* Intuitively we know that if you leave late you will probably arrive late, so we can easily see why the amount of time delayed before the scheduled departure time will contain the most (in the model it looks like almost all) the information we need to predict if a flight will be 30 minutes or more late.

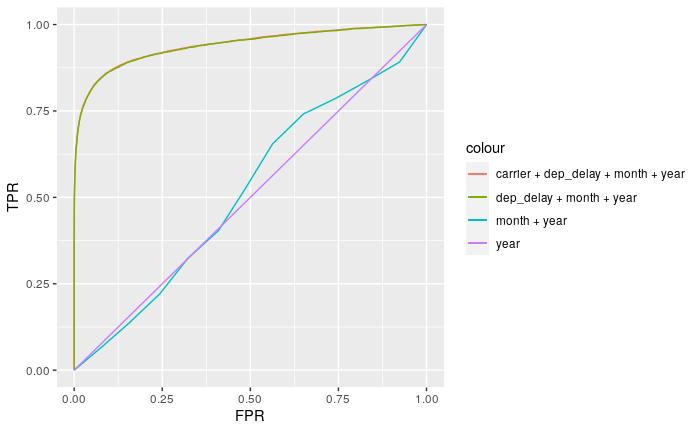
****

Figure 4. ROC curve comparing different models.

# Major assignment 2: Part 3

1. (a) Load the wine dataset directly from GitHub using the command provided
2. Change the Wine variable to categorical.

[1 point]

Your input code and output code go here:

**Load the dataset with the provided code**

library(readr)

library(dplyr)

library(caret)

url <- "https://gist.githubusercontent.com/tijptjik/9408623/raw/b237fa5848349a14a14e5d4107dc7897c21951f5/wine.csv"

wine\_df <-  read\_delim(url, delim = ",")

**Change the Wine variable to categorical**

wine\_df$Wine <- factor(wine\_df$Wine)

**Verify the variable change**

wine\_df

Table 6. Instead of having the original <dbl> datatype, Wine is now shown as <fct>.

|  |
| --- |
| # A tibble: 178 x 14  Wine Alcohol Malic.acid Ash Acl Mg Phenols Flavanoids Nonflavanoid.phenols Proanth Color.int Hue OD Proline  <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  1 1 14.2 1.71 2.43 15.6 127 2.8 3.06 0.28 2.29 5.64 1.04 3.92 1065  2 1 13.2 1.78 2.14 11.2 100 2.65 2.76 0.26 1.28 4.38 1.05 3.4 1050  3 1 13.2 2.36 2.67 18.6 101 2.8 3.24 0.3 2.81 5.68 1.03 3.17 1185  4 1 14.4 1.95 2.5 16.8 113 3.85 3.49 0.24 2.18 7.8 0.86 3.45 1480  5 1 13.2 2.59 2.87 21 118 2.8 2.69 0.39 1.82 4.32 1.04 2.93 735  6 1 14.2 1.76 2.45 15.2 112 3.27 3.39 0.34 1.97 6.75 1.05 2.85 1450  7 1 14.4 1.87 2.45 14.6 96 2.5 2.52 0.3 1.98 5.25 1.02 3.58 1290  8 1 14.1 2.15 2.61 17.6 121 2.6 2.51 0.31 1.25 5.05 1.06 3.58 1295  9 1 14.8 1.64 2.17 14 97 2.8 2.98 0.29 1.98 5.2 1.08 2.85 1045  10 1 13.9 1.35 2.27 16 98 2.98 3.15 0.22 1.85 7.22 1.01 3.55 1045 |

1. (a) Set the random number seed to 42 to cause reproducible training.
2. Use the caret and nnet package to classify Wine based on all other attributes, using default training/structure.

[2 points]

Your input and output code goes here:

# set the random seed

set.seed(42)

# create the model

model <- train(Wine ~ ., wine\_df, method='nnet', trace=FALSE)

1. Plot the trained network. [1 point]

Your input and output code goes here:

# load the library and r code for plotting a neural net

library(devtools)

source\_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a505ff044412703516c34f1a4684a5/nnet\_plot\_update.r')

# plot the model

plot.nnet(model,nid=T,var.labs=F)

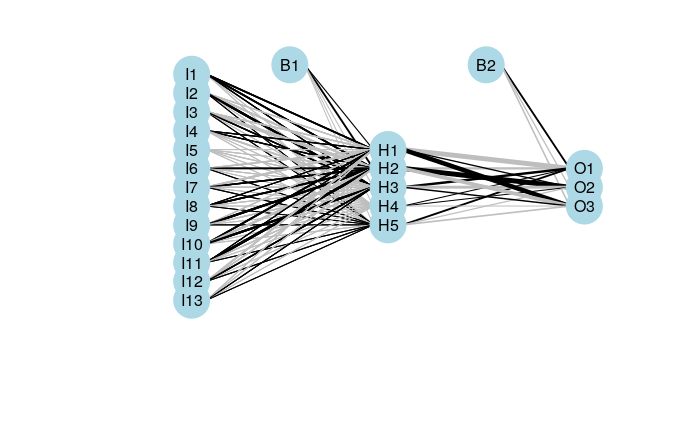


Figure 5. Plot of the trained neural net.

1. Predict the Wine variable from the trained network and produce a classification table of predicted vs actual Wine value/category. [2 points]

Your input and output code goes here:

prediction <- predict(model, wine\_df)

table(prediction, wine\_df$Wine)

Table 7. Wine model Confusion Matrix.

|  |
| --- |
| prediction 1 2 3  1 57 0 0  2 2 70 0  3 0 1 48 |

# Major assignment 2: Part 4

1. Initiate a H2O session [1 point]

Your input code and output code go here:

library(nycflights13)

library(h2o)

h2o.init()

Table 8. H2O session details.

|  |
| --- |
| Connection successful!  R is connected to the H2O cluster:  H2O cluster uptime: 2 hours 12 seconds  H2O cluster timezone: Etc/UTC  H2O data parsing timezone: UTC  H2O cluster version: 3.32.1.4  H2O cluster version age: 24 days  H2O cluster name: H2O\_started\_from\_R\_rstudio\_hig442  H2O cluster total nodes: 1  H2O cluster total memory: 3.01 GB  H2O cluster total cores: 4  H2O cluster allowed cores: 4  H2O cluster healthy: TRUE  H2O Connection ip: localhost  H2O Connection port: 54321  H2O Connection proxy: NA  H2O Internal Security: FALSE  H2O API Extensions: Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder, Core V4  R Version: R version 4.0.5 (2021-03-31) |

1. Load the nycflights13 data as a data frame. [1 point]

Your input and output code goes here:

flights <- nycflights13::flights

1. Write the data out to a .csv file and read it back again (this solves a data format issue). [1 point]

Your input and output code goes here:

write.csv(flights,"flights.csv", row.names=FALSE)

flights <- read.csv("flights.csv")

1. Create a H2O data frame from the data. [1 point]

Your input and output code goes here:

flights.hex <- as.h2o(flights)

h2o.describe(flights.hex)

Table 9. H2O Data frame.

|  |
| --- |
| Label Type Missing Zeros PosInf NegInf Min Max Mean Sigma Cardinality  1 year int 0 0 0 0 2013 2013 2013.000000 0.000000 NA  2 month int 0 0 0 0 1 12 6.548510 3.414457 NA  3 day int 0 0 0 0 1 31 15.710787 8.768607 NA  4 dep\_time int 8255 0 0 0 1 2400 1349.109947 488.281791 NA  5 sched\_dep\_time int 0 0 0 0 106 2359 1344.254840 467.335756 NA  6 dep\_delay int 8255 16514 0 0 -43 1301 12.639070 40.210061 NA  7 arr\_time int 8713 0 0 0 1 2400 1502.054999 533.264132 NA  8 sched\_arr\_time int 0 0 0 0 1 2359 1536.380220 497.457142 NA  9 arr\_delay int 9430 5409 0 0 -86 1272 6.895377 44.633292 NA  10 carrier string 0 0 0 0 NaN NaN NA NA NA  11 flight int 0 0 0 0 1 8500 1971.923620 1632.471938 NA  12 tailnum string 2512 0 0 0 NaN NaN NA NA NA  13 origin string 0 0 0 0 NaN NaN NA NA NA  14 dest string 0 0 0 0 NaN NaN NA NA NA  15 air\_time int 9430 0 0 0 20 695 150.686460 93.688305 NA  16 distance int 0 0 0 0 17 4983 1039.912604 733.233033 NA  17 hour int 0 0 0 0 1 23 13.180247 4.661316 NA  18 minute int 0 60696 0 0 0 59 26.230100 19.300846 NA  19 time\_hour string 0 0 0 0 NaN NaN NA NA NA |

1. Create a new categorical variable in the frame called late\_arrival that is 1 if arr\_delay is greater than or equal to 30. [1 point]

Your input and output code goes here:

flights.hex$late\_arrival <- as.factor(flights.hex$arr\_delay >= 30)

h2o.describe(flights.hex)

Table 10. Addition of late\_arrival variable (20).

|  |
| --- |
| Label Type Missing Zeros PosInf NegInf Min Max Mean Sigma Cardinality  1 year int 0 0 0 0 2013 2013 2013.0000000 0.0000000 NA  2 month int 0 0 0 0 1 12 6.5485100 3.4144572 NA  3 day int 0 0 0 0 1 31 15.7107870 8.7686071 NA  4 dep\_time int 8255 0 0 0 1 2400 1349.1099473 488.2817910 NA  5 sched\_dep\_time int 0 0 0 0 106 2359 1344.2548400 467.3357557 NA  6 dep\_delay int 8255 16514 0 0 -43 1301 12.6390703 40.2100609 NA  7 arr\_time int 8713 0 0 0 1 2400 1502.0549986 533.2641320 NA  8 sched\_arr\_time int 0 0 0 0 1 2359 1536.3802201 497.4571415 NA  9 arr\_delay int 9430 5409 0 0 -86 1272 6.8953768 44.6332917 NA  10 carrier string 0 0 0 0 NaN NaN NA NA NA  11 flight int 0 0 0 0 1 8500 1971.9236199 1632.4719381 NA  12 tailnum string 2512 0 0 0 NaN NaN NA NA NA  13 origin string 0 0 0 0 NaN NaN NA NA NA  14 dest string 0 0 0 0 NaN NaN NA NA NA  15 air\_time int 9430 0 0 0 20 695 150.6864602 93.6883047 NA  16 distance int 0 0 0 0 17 4983 1039.9126036 733.2330333 NA  17 hour int 0 0 0 0 1 23 13.1802474 4.6613157 NA  18 minute int 0 60696 0 0 0 59 26.2300995 19.3008457 NA  19 time\_hour string 0 0 0 0 NaN NaN NA NA NA  20 late\_arrival enum 0 283974 0 0 0 1 0.1567867 0.3636001 2 |

1. Create a data split of the H2O frame with 80% of the data being used for training (flight\_train) and 20% used for testing (flight\_test), ensuring repeatability by setting the seed for the random sampling of the split to be 42. [1 point]

Your input and output code goes here:

splits <- h2o.splitFrame(data = flights.hex,

                         ratios = c(0.8),

                         seed = 42)

flight\_train <- splits[[1]]

flight\_test <- splits[[2]]

print(paste0("Number of rows in train set: ", h2o.nrow(flight\_train)))

print(paste0("Number of rows in test set: ", h2o.nrow(flight\_test)))

Table 11. Flights data split.

|  |
| --- |
| [1] "Number of rows in train set: 269572"  [1] "Number of rows in test set: 67204" |

1. Create a predictors set *predictors* from sched\_dep\_time, dep\_delay, air\_time, and distance. [1 point]

Your input and output code goes here:

predictors = c("sched\_dep\_time", "dep\_delay", "air\_time", "distance")

1. **Create a deep learning H20 model using the defaults except for:**
   * **the predictors being predictors,**
   * **the response variable being late\_arrival,**
   * **the training data being flight\_train, and**
   * **the results forced to be reproducible with seed 42.**

Run several times to check your results are reproducible.  
[2 points]

Your input and output code goes here:

flights.dl <- h2o.deeplearning(x=predictors, y=c("late\_arrival"), training\_frame=flight\_train, seed=42)

summary(flights.dl)

Table 12. Model Summary.

|  |
| --- |
| Model Details:  ==============  H2OBinomialModel: deeplearning  Model Key: DeepLearning\_model\_R\_1627921350458\_1  Status of Neuron Layers: predicting late\_arrival, 2-class classification, bernoulli distribution, CrossEntropy loss, 41,602 weights/biases, 496.0 KB, 1,592,367 training samples, mini-batch size 1  layer units type dropout l1 l2 mean\_rate rate\_rms momentum mean\_weight weight\_rms mean\_bias bias\_rms  1 1 4 Input 0.00 % NA NA NA NA NA NA NA NA NA  2 2 200 Rectifier 0.00 % 0.000000 0.000000 0.018740 0.097991 0.000000 -0.027241 0.204710 -0.081594 0.105511  3 3 200 Rectifier 0.00 % 0.000000 0.000000 0.408141 0.378315 0.000000 -0.041165 0.124046 -0.033537 0.395143  4 4 2 Softmax NA 0.000000 0.000000 0.026071 0.067706 0.000000 0.021736 0.363663 -0.004755 1.071725  H2OBinomialMetrics: deeplearning  \*\* Reported on training data. \*\*  \*\* Metrics reported on temporary training frame with 9977 samples \*\*  MSE: 0.03819632  RMSE: 0.1954388  LogLoss: 0.1399449  Mean Per-Class Error: 0.1162488  AUC: 0.9664594  AUCPR: 0.9152095  Gini: 0.9329187  Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  0 1 Error Rate  0 8251 116 0.013864 =116/8367  1 352 1258 0.218634 =352/1610  Totals 8603 1374 0.046908 =468/9977  Maximum Metrics: Maximum metrics at their respective thresholds  metric threshold value idx  1 max f1 0.599637 0.843164 126  2 max f2 0.186248 0.848787 247  3 max f0point5 0.804351 0.898571 79  4 max accuracy 0.651344 0.953693 112  5 max precision 0.999772 1.000000 0  6 max recall 0.001057 1.000000 396  7 max specificity 0.999772 1.000000 0  8 max absolute\_mcc 0.651344 0.820985 112  9 max min\_per\_class\_accuracy 0.133725 0.907255 271  10 max mean\_per\_class\_accuracy 0.186248 0.910970 247  11 max tns 0.999772 8367.000000 0  12 max fns 0.999772 1502.000000 0  13 max fps 0.000121 8367.000000 399  14 max tps 0.001057 1610.000000 396  15 max tnr 0.999772 1.000000 0  16 max fnr 0.999772 0.932919 0  17 max fpr 0.000121 1.000000 399  18 max tpr 0.001057 1.000000 396  Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)`  Scoring History:  timestamp duration training\_speed epochs iterations samples training\_rmse training\_logloss  1 2021-08-02 16:26:27 0.000 sec NA 0.00000 0 0.000000 NA NA  2 2021-08-02 16:26:42 15.964 sec 3373 obs/sec 0.17788 1 47951.000000 0.26842 0.25483  3 2021-08-02 16:27:00 33.537 sec 4609 obs/sec 0.53470 3 144139.000000 0.20920 0.16106  4 2021-08-02 16:27:09 43.090 sec 6058 obs/sec 0.89470 5 241186.000000 0.19731 0.14826  5 2021-08-02 16:27:25 58.828 sec 6141 obs/sec 1.25043 7 337080.000000 0.19894 0.15109  6 2021-08-02 16:27:35 1 min 8.933 sec 6736 obs/sec 1.60956 9 433891.000000 0.19580 0.14182  7 2021-08-02 16:27:44 1 min 18.064 sec 7272 obs/sec 1.96743 11 530364.000000 0.19706 0.14746  8 2021-08-02 16:27:51 1 min 24.758 sec 7333 obs/sec 2.14719 12 578821.000000 0.19593 0.14276  9 2021-08-02 16:28:02 1 min 35.995 sec 7541 obs/sec 2.50683 14 675772.000000 0.19886 0.15010  10 2021-08-02 16:28:08 1 min 41.970 sec 7634 obs/sec 2.68601 15 724073.000000 0.20197 0.14949  11 2021-08-02 16:28:17 1 min 51.294 sec 7904 obs/sec 3.04378 17 820517.000000 0.19467 0.14042  12 2021-08-02 16:28:24 1 min 58.271 sec 8304 obs/sec 3.39999 19 916541.000000 0.19730 0.14849  13 2021-08-02 16:28:31 2 min 4.847 sec 8691 obs/sec 3.75730 21 1012864.000000 0.19544 0.13994  14 2021-08-02 16:28:38 2 min 11.625 sec 9019 obs/sec 4.11512 23 1109322.000000 0.19662 0.14257  15 2021-08-02 16:28:44 2 min 18.216 sec 9327 obs/sec 4.47267 25 1205706.000000 0.21672 0.16988  16 2021-08-02 16:28:50 2 min 24.144 sec 9654 obs/sec 4.83121 27 1302358.000000 0.19486 0.14078  17 2021-08-02 16:28:57 2 min 31.180 sec 9889 obs/sec 5.18906 29 1398824.000000 0.21326 0.16533  18 2021-08-02 16:29:06 2 min 40.012 sec 9984 obs/sec 5.54810 31 1495613.000000 0.19586 0.14247  19 2021-08-02 16:29:14 2 min 47.650 sec 10140 obs/sec 5.90702 33 1592367.000000 0.19775 0.14369  20 2021-08-02 16:29:14 2 min 48.109 sec 10138 obs/sec 5.90702 33 1592367.000000 0.19544 0.13994  training\_r2 training\_auc training\_pr\_auc training\_lift training\_classification\_error  1 NA NA NA NA NA  2 0.46762 0.95945 0.89868 6.13493 0.05002  3 0.67661 0.96423 0.90999 6.19689 0.04921  4 0.71233 0.96380 0.90943 6.13493 0.04931  5 0.70755 0.96572 0.91224 6.13493 0.04711  6 0.71670 0.96545 0.91130 6.13493 0.04721  7 0.71304 0.96470 0.91052 6.13493 0.04681  8 0.71634 0.96523 0.91175 6.19689 0.04871  9 0.70779 0.96530 0.91149 6.13493 0.04681  10 0.69859 0.96619 0.91212 6.19689 0.04871  11 0.71997 0.96666 0.91510 6.19689 0.04831  12 0.71236 0.96514 0.91067 6.19689 0.04831  13 0.71776 0.96646 0.91521 6.19689 0.04691  14 0.71432 0.96541 0.91400 6.19689 0.04871  15 0.65294 0.96589 0.91406 6.19689 0.04721  16 0.71941 0.96484 0.91246 6.19689 0.04721  17 0.66393 0.96161 0.90354 6.19689 0.05382  18 0.71653 0.96568 0.91435 6.19689 0.04741  19 0.71105 0.96453 0.91295 6.19689 0.04801  20 0.71776 0.96646 0.91521 6.19689 0.04691  Variable Importances: (Extract with `h2o.varimp`)  =================================================  Variable Importances:  variable relative\_importance scaled\_importance percentage  1 air\_time 1.000000 1.000000 0.320502  2 dep\_delay 0.932594 0.932594 0.298898  3 distance 0.898379 0.898379 0.287932  4 sched\_dep\_time 0.289134 0.289134 0.092668 |

1. Plot an ROC curve for your model using the flight\_test data. [1 point]

Your input and output code goes here:

flights.perf <- h2o.performance(model=flights.dl, newdata=flight\_test)

plot(flights.perf, type="roc")

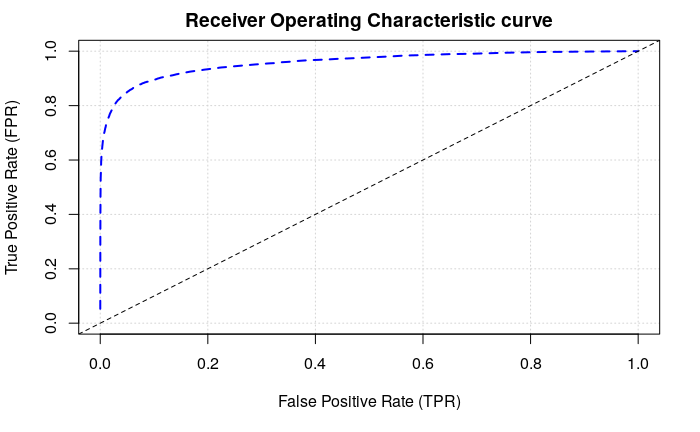


Figure 6. ROC curve for the test dataset.

1. Calculate the prediction accuracy produced by your model using h2o.predict on your model and test data flight\_test. [1 point]

Your input and output code goes here:

**Perform Predictions**

pred <- h2o.predict(flights.dl, newdata=flight\_test)

**Model Performance**

flights.perf

Table 13. Model metrics based on the test dataset.

|  |
| --- |
| H2OBinomialMetrics: deeplearning  MSE: 0.04022622  RMSE: 0.2005648  LogLoss: 0.1512332  Mean Per-Class Error: 0.1163139  AUC: 0.9579824  AUCPR: 0.8971274  Gini: 0.9159649  Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  0 1 Error Rate  0 55439 1312 0.023119 =1312/56751  1 2190 8263 0.209509 =2190/10453  Totals 57629 9575 0.052110 =3502/67204  Maximum Metrics: Maximum metrics at their respective thresholds  metric threshold value idx  1 max f1 0.439108 0.825145 171  2 max f2 0.173685 0.832861 257  3 max f0point5 0.825695 0.884503 72  4 max accuracy 0.527455 0.949080 147  5 max precision 0.995112 0.996930 6  6 max recall 0.000135 1.000000 399  7 max specificity 0.999763 0.999930 0  8 max absolute\_mcc 0.527455 0.797585 147  9 max min\_per\_class\_accuracy 0.110935 0.896106 288  10 max mean\_per\_class\_accuracy 0.171377 0.902593 258  11 max tns 0.999763 56747.000000 0  12 max fns 0.999763 9898.000000 0  13 max fps 0.000135 56751.000000 399  14 max tps 0.000135 10453.000000 399  15 max tnr 0.999763 0.999930 0  16 max fnr 0.999763 0.946905 0  17 max fpr 0.000135 1.000000 399  18 max tpr 0.000135 1.000000 399  Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)` |

**The result is calculated using the error rate provided in the confusion matrix**

> 1 - 0.052110

[1] 0.94789

1. Calculate the prediction accuracy that would be yielded, for the test data flight\_test, by simply using dep\_delay (being larger than 30) alone as a predictor. [1 point]

Your input and output code goes here:

flights.hex$late\_departure <- as.factor(flights.hex$dep\_delay > 30)

splits <- h2o.splitFrame(data = flights.hex,

                         ratios = c(0.8),

                         seed = 42)

flight\_train <- splits[[1]]

flight\_test <- splits[[2]]

flights.dl <- h2o.deeplearning(x=c("late\_departure"), y=c("late\_arrival"), training\_frame=flight\_train, seed=42)

flights.perf <- h2o.performance(model=flights.dl, newdata=flight\_test)

flights.perf

Table 14. Performance using late departure feature only.

|  |
| --- |
| H2OBinomialMetrics: deeplearning  MSE: 0.05436655  RMSE: 0.2331664  LogLoss: 0.2143271  Mean Per-Class Error: 0.13123  AUC: 0.86877  AUCPR: 0.7110713  Gini: 0.7375399  Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:  0 1 Error Rate  0 55201 1550 0.027312 =1550/56751  1 2458 7995 0.235148 =2458/10453  Totals 57659 9545 0.059639 =4008/67204  Maximum Metrics: Maximum metrics at their respective thresholds  metric threshold value idx  1 max f1 0.852224 0.799580 0  2 max f2 0.852224 0.778375 0  3 max f0point5 0.852224 0.821973 0  4 max accuracy 0.852224 0.940361 0  5 max precision 0.852224 0.837611 0  6 max recall 0.043935 1.000000 1  7 max specificity 0.852224 0.972688 0  8 max absolute\_mcc 0.852224 0.765722 0  9 max min\_per\_class\_accuracy 0.852224 0.764852 0  10 max mean\_per\_class\_accuracy 0.852224 0.868770 0  11 max tns 0.852224 55201.000000 0  12 max fns 0.852224 2458.000000 0  13 max fps 0.043935 56751.000000 1  14 max tps 0.043935 10453.000000 1  15 max tnr 0.852224 0.972688 0  16 max fnr 0.852224 0.235148 0  17 max fpr 0.043935 1.000000 1  18 max tpr 0.043935 1.000000 1  Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>, xval=<T/F>)` |

We can use the error rate in the confusion matrix to calculate the model accuracy:

> (1 - 0.059639) \* 100

[1] 94.0361

Subtracting the error rate of “full” model (0.052110) from the present model (0.059639) and multiplying by 100 will show the difference (improvement) as a percentage.

> (0.059639 - 0.052110) \* 100

[1] 0.7529