STAT 652
Project Report
Neda Zolaktaf – 301374382
December 6, 2019

1. Introduction

The goal of this project is to study the nycflights13 package. This package contains information about all flights which have departed from NYC to destinations in the United States, Puerto Rico, and the American Virgin Islands in 2013. The dataset is gathered from the following datasets which can help users understand the causes of delays better [1]:

- The flights dataset, which consists of all flights that departed from NYC in 2013.
- The weather dataset, which consists of hourly meterological data for each airport.
- The airports dataset, which consists of airport names and locations.
- The planes dataset, which consists of construction information about each plane.

The number of flights is 336,776 in total. The dataset contains 200000 rows and 43 features. See Section 2 for more details about the dataset.

In this project, our main task is to build prediction models for departure delays, which is given as dep_delay in the dataset, based on the other 42 features. This task can be either solved as a binary classification problem or as a regression problem (continuous value). Therefore, I consider both of the approaches. For the classification problem, I calculate the median of the departure delays to determine the labels of the flights. If the dep_delay is higher than the median for a flight, then I consider it as a delayed flight.

The dataset is divided to train and validation set, so we can use the validation set for tuning different models. After training the machine learning models a test set is provided which enables us to calculate the error of each tuned model. The evaluation metric in the binary classification models is the misclassification error and the evaluation metric for regression models is a normalized version of the Mean Squared Error (MSE).

2. Dataset

As mentioned above, the package contains the flights, weather, airports and the planes datasets. I will explain the features of each dataset in the following:

- The flight dataset contains on-time data for all flights that departed NYC which contains 336776 rows and 19 columns. The columns are flights, year, month, day, dep_time, arr_time, dep_delay, arr_delay, flight, talinum, origin, dest, air_time, distance, hour, minute, time_hour.
- The weather dataset contains hourly meterological data for LGA, JFK and EWR. The dataset contains 26115 rows and 15 features which are origin, year, month, day, hour, temp, dewp, wind_dir, wind_speed, wind_gust, precip, pressure, visib, time_hour.
- The airport dataset contains Useful metadata about airports which has 1458 rows and 8 columns. The columns are faa, name, lat, lon, alt, tz, dst, tzone.
- The planes dataset has 3322 rows and 9 columns. The dataset is plane metadata for all plane tail numbers found in the FAA aircraft registry. American Airways (AA) and Envoy Air (MQ) can't

be matched because they report fleet numbers rather than tail numbers. The columns of this dataset include talinum, year, type, manufacturer, model, engines, seats, speed, engine.

3. Methods

The first step in working with every dataset is preprocessing. The dataset contains some missing values which have to be either imputed or removed. For example, one rule is to discard variables with more than 5% missing values. Another important step is mapping extremes to something less extreme. We can map extremes to quantiles of the standard normal or to rank. Another important step is removing some highly correlated variables or converting some variables to other types.

After preprocessing the data and removing the highly correlated variables the training set is divided to a train and validation set in order to tune the model. After preprocessing, the train set contains 122878 rows and 17 columns and the validation set contains 61438 rows and 17 columns. For the test set, I execute the same preprocessing methods on the dataset. There was an unexpected value in the test set which I had to remove: factor dest has new levels LEX. So I added this code in test set preprocessing:
fl_te_tem_class_test<-fl_te_tem_class_test[!(fl_te_tem_class_test$dest=="LEX"),]

3.1 First Approach (Classification)

As I previously mentioned there are two main approaches for predicting the departure delays. In the first approach (classification), I use a categorical variable for the output. I use the median as a threshold to determine the label. If the value of the dep_delay is bigger than the threshold then the label is True (so it is classified as a delayed flight) and if the value is less than the threshold then the label is False.
I used the median of dep_delay as the threshold not the mean, since the mean can be affected by the outliers and it would be an inappropriate threshold (meaningless).

3.1.1 Predictive Models

Logistic Regression: Logistic regression could be used when there are only two (binary) possible labels. This method is used to explain the relationship between the predictor and other variable. Logistic regression could also be extended to multi-class classification by changing the objective function, which is usually called multinomial logistic regression.

Linear Discriminant Analysis (LDA): LDA is a dimensionality reduction technique. It is mostly used for supervised classification problems (classification problems were the output variable is categorical). LDA supports multi-class classification as well.
The extensions of LDA include Quadratic Discriminant Analysis (QDA), Flexible Discriminant Analysis (FDA), Regularized Discriminant Analysis (RDA).

Support Vector Machine (SVM) (Linear Kernel): SVM is a supervised machine learning algorithm which performs classification by a hyperplane that maximizes the margin between two classes. SVM can be used in both classification and regression problems.
SVM uses a set of mathematical functions that are defined as kernels, such as the Radial Basis Function kernel, for when the data is not linearly separable. The linear kernel works fine if your dataset if linearly separable [2]. Given an arbitrary dataset, you typically do not know which kernel may work best and can test all kernels to choose the best one.

SVM (Radial Kernel): Nonlinear kernels, such as the Radial Basis Function kernel, are used for non-linear problems. I use both kernels for making the prediction. However, it was taking a very long time for the regression problem to be executed, so I excluded it from the comparison but the code is in the Appendix.

Boosting: Boosting is an algorithm which converts a weak learner to a strong learner. It is an ensemble method for improving the model predictions of any given learning algorithm. The idea of boosting is to train weak learners sequentially in order to correct its' predecessor [4]. Boosting could be used for both classification and regression problems. For regression, we change the distribution parameter to Gaussian.

Boosting with default shrinkage: A shrinkage parameter is applied to each tree, it is also known as learning rate. The Default value is 0.1. Smaller learning rate typically requires more trees.

Random Forest: Random Forest is a supervised learning algorithm which is used for both regression and classification. We can tune the number of variables randomly sampled at each split. I used this model as my main method and tuned it on the validation set to minimize the error on the test set.

Generalized Additive Models (GAM): GAM models relax the restriction that the relationship between features must be a simple weighted sum. The outcome could be modeled by a sum of arbitrary functions of each feature.

3.1.2 Evaluation Metric

To evaluate the models for the classification problem, I used the misclassification error, which is equal to (Number of misclassified predictions / Total number of predictions).

3.2 Second Approach (Regression)

3.2.1 Predictive Models

Boosting: As I explained earlier boosting could be used for either Classification or Regression problems. For classification we change the distribution parameter to Bernoulli.

Random Forest: In order to get the best model I tuned the number of trees and the number of variables randomly sampled as candidates at each split (mtry).

Generalized Additive Models (GAM): This method could be used in either classification or regression problems.

Support Vector Machine (SVM): SVM can be used in both classification and regression challenge. Since it was taking a long time for my code to execute I just added the code in the appendix section.

Lasso Regression: Lasso regression is a linear regression that uses shrinkage. It performs L1 regularization which adds a penalty to regularize the weights and avoid overfitting the train data. λ is the amount of shrinkage which controls the strength of the penalty [5].

Ridge Regression: Ridge regression uses L2 regularization technique to regularize the weights and avoid overfitting the train data. You can control the penalty term by changing alpha. The bigger the alpha is the magnitude of coefficients are reduced [6].

3.2.2 Evaluation Metric

To evaluate the models for regression, I used a normalized version of Mean Squared Error (MSE). MSE is the average squared difference between the estimated values and the actual value. However, we use a baseline to compare the methods to which is var_dd <- var(fl_te$dep_delay). So the evaluation metric for all models is abs(mse - var_dd)/var_dd.

4. Results

Table 1 demonstrates the misclassification error for classification models. Table 2 demonstrates the normalized MSE for regression models.

Model	Validation set	Test set
Logistic Regression	0.3723103	0.3714265
LDA	0.6279339	0.6287322
SVM (Radial Kernel)	0.3601029	Took a long time for model to run
SVM (linear Kernel)	0.3710733	Took a long time for model to run
Boosting	0.499235	0.4958821
Random Forest	0.3693805	0.3668772
Naïve Bayes	0.9999837	0.4015266
GAM	0.3585891	0.3584458

Table 1: Misclassification error for classification models.

Model	Validation set	Test set
Boosting	0.1329642	0.1364254
Random Forest	0.1601977	0.165758
GAM	0.1521285	0.1566455
Boosting without shrinkage	0.1596607	0.163518
Lasso Regression	0.1218238	0.1273962
Ridge Regression	0.03269137	0.0333249

Table 2: Normalized MSE for regression models.

4.1 Best Model Among Classification Models

- GAM and Random Forest perform the best among the other models.
- LDA and Naïve Bayes perform the worst.

4.2 Best Model Among Regression Models

- Ridge Regression outperforms the other models.
- Random Forest performs the worst among all models.

5. Conclusion

5.1 Summary

In this project, our main task is to build prediction models for departure delays which is given as dep_delay in the dataset. As the first step, I gained insight into the problem by exploring the data. I addressed the problem by modeling it as both classification and regression problems. In both of the approaches, I considered various models to make a prediction on the dept_delay variable. As mentioned earlier, in the classification problem, all methods have a good performance. However, GAM and Random Forest perform better compared to other models. For the regression problem, Ridge Regression outperforms the other methods. For the SVM model we can tune the parameters, but we had a time limitation in this project and it can be considered as a future work.

5.2 Future work

- Consider only the features which had non-zero weights in lasso and ridge regression and use them as input features in other models.
- Consider polynomial logistic regression instead of simple logistic regression.
- Consider the extension of LDA such as Quadratic Discriminant Analysis where each class uses its own estimate of variance or Regularized Discriminant Analysis which introduces regularization into the estimate of the variance [3].

5.3 References

[1] https://github.com/hadley/nycflights13
[2] https://www.kdnuggets.com/2016/06/select-support-vector-machine-kernels.html
[3] https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/
[4] https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5
[5] https://www.statisticshowto.datasciencecentral.com/lasso-regression/
[6] https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/

APPENDIX_ R CODE

Statistics 452: Statistical Learning and Prediction

Neda Zolaktaf

R version

```
knitr::opts_chunk$set(echo = TRUE)
R.version
```

```
##                _
## platform       x86_64-apple-darwin15.6.0
## arch           x86_64
## os             darwin15.6.0
## system         x86_64, darwin15.6.0
## status
## major          3
## minor          6.1
## year           2019
## month          07
## day            05
## svn rev        76782
## language       R
## version.string R version 3.6.1 (2019-07-05)
## nickname       Action of the Toes
```

Loading the required libraries

```
library(lubridate)
library(randomForest)
library(gbm)
library(dplyr)
library(leaps)
library(glmnet)
library(dplyr)
library(gam)
library(e1071)
library(MASS)
```

Flights dataset

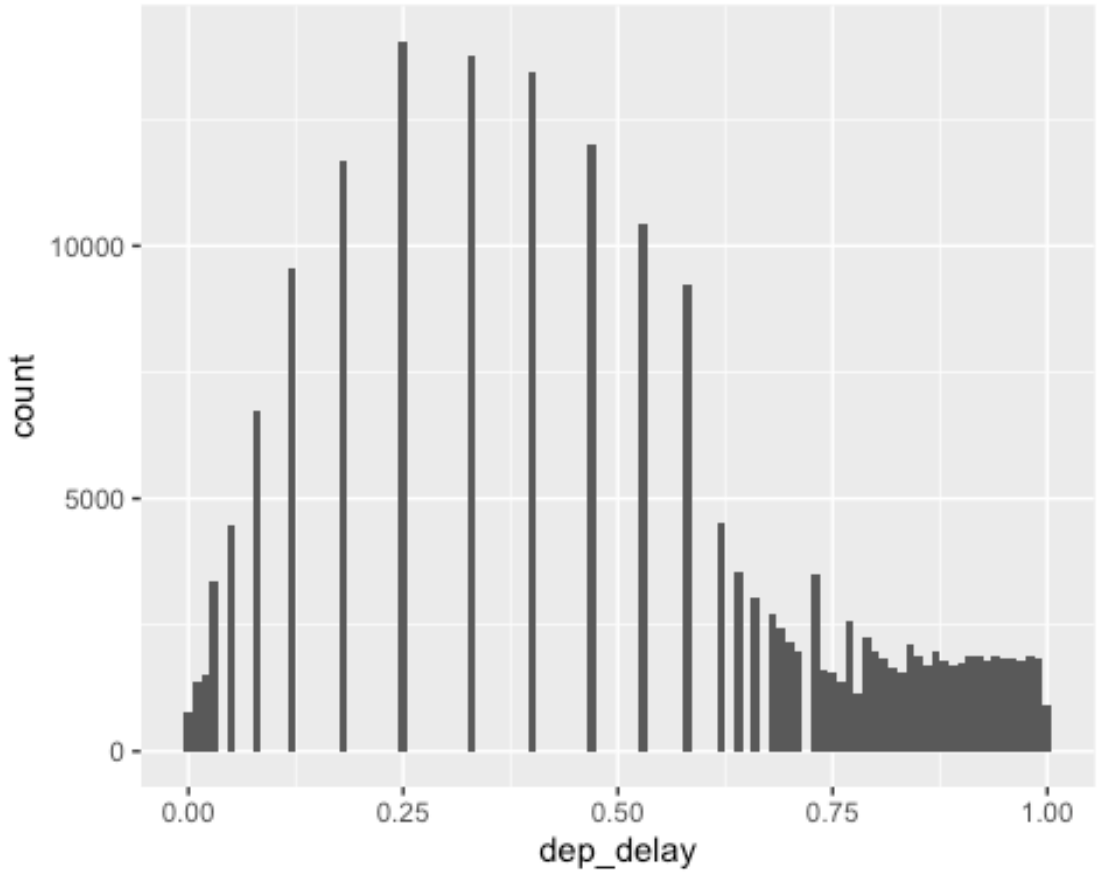
```r
library(tidyverse)
library(nycflights13)
#help(flights)
#help(weather)
#help(airports)
#help(planes)
fltrain <- read_csv("fltrain.csv.gz")
fltrain
```

Preprocessing (Professors Code)

```r
fl <- fltrain
for(i in 1:ncol(fl)) {
  if(typeof(fl[[i]]) == "character") {
    fl[[i]] <- factor(fl[[i]])
  }
}


num_miss <- function(x) { sum(is.na(x)) }
sapply(fl,num_miss)

fl <- fl%>%
  select(-year.y,-type,-manufacturer,-model,-engines,-seats, -speed, -engine,
-wind_gust,-pressure)


fl <- na.omit(fl)
summary(fl)

range(fl$dep_delay)

## [1]  -43 1301

fivenum(fl$dep_delay)

## [1]  -43   -5   -2   11 1301

quantile(fl$dep_delay,probs = c(0.01,0.05,0.1,0.25,.5,.75,.90,.95,.99))

##  1%  5% 10% 25% 50% 75% 90% 95% 99%
## -12  -9  -7  -5  -2  11  49  88 193

mean(fl$dep_delay >= 60) # about 15,000 or 8% of flights

## [1] 0.08210356

fl%>% arrange(desc(dep_delay)) %>% head(10)

Q3 <- function(x) { quantile(x,probs=.75) }
fl %>% group_by(origin) %>%
  summarize(n=n(),med_d = median(dep_delay),Q3_d = Q3(dep_delay), max_d = max
```

```
(dep_delay)) %>%
  arrange(desc(Q3_d)) %>% head(10)

## # A tibble: 3 x 5
##   origin     n med_d  Q3_d max_d
##   <fct>  <int> <dbl> <dbl> <dbl>
## 1 EWR    65512    -1    15   896
## 2 JFK    60327    -1    10  1301
## 3 LGA    58477    -3     7   911

fl %>% group_by(carrier) %>%
  summarize(n=n(),med_d = median(dep_delay),Q3_d = Q3(dep_delay), max_d = max
(dep_delay)) %>%
  arrange(desc(Q3_d)) %>% head(10)

fl %>% group_by(origin,carrier) %>%
  summarize(n=n(),med_d = median(dep_delay),Q3_d = Q3(dep_delay), max_d = max
(dep_delay)) %>%


fl %>% group_by(dest,carrier) %>%
  summarize(n=n(),med_d = median(dep_delay),Q3_d = Q3(dep_delay), max_d = max
(dep_delay)) %>%
  arrange(desc(Q3_d)) %>% head(10)

## # A tibble: 10 x 6
## # Groups:   dest [10]
##    dest  carrier     n med_d  Q3_d max_d
##    <fct> <fct>   <int> <dbl> <dbl> <dbl>
##  1 STL   UA          2  77.5 116.    155
##  2 DTW   OO          2  61    96     131
##  3 TYS   EV        183   8    68.5   285
##  4 PBI   EV          3  50    67.5    85
##  5 ORD   OO          1  67    67      67
##  6 RDU   UA          1  60    60      60
##  7 TUL   EV        185   3    53     251
##  8 OKC   EV        184   8.5  51.5   207
##  9 BHM   EV        175   3    50     325
## 10 CAE   EV         57  10    48     163

fl %>% group_by(month,day) %>%
  summarize(n=n(),med_d = mean(dep_delay),max_d = max(dep_delay)) %>%
  arrange(desc(med_d)) %>% head(10) # what happened on march 8?



fl %>% mutate(haveprecip = factor(precip>0)) %>% group_by(haveprecip) %>%
  summarize(n=n(),med_d = median(dep_delay),Q3_d = Q3(dep_delay), max_d = max
(dep_delay)) %>%
  arrange(desc(med_d)) %>% head(10)
```

```
## # A tibble: 2 x 5
##   haveprecip      n med_d  Q3_d max_d
##   <fct>       <int> <dbl> <dbl> <dbl>
## 1 TRUE        11804     5    41   853
## 2 FALSE      172512    -2     9  1301

den <- nrow(fl)+1
fl <- fl %>% mutate(dep_delay = rank(dep_delay)/den)
 ggplot(fl,aes(x=dep_delay)) + geom_histogram(binwidth=.01)
```




```
 library(lubridate)
fl <- fl %>%
  mutate(dep_date = make_date(year.x,month,day)) %>%
  select(-year.x,-month,-day,-dep_time,-arr_time,-arr_delay,
         -sched_arr_time,-tailnum,-flight,-name,-air_time,
         -hour,-minute,-time_hour,-tz,-dst,-tzone) %>%
  mutate(precip = as.numeric(precip>0))

ggplot(fl,aes(x=dep_date,y=dep_delay)) + geom_point(alpha=.01) + geom_smooth(
)
```

```
# Definitely non-linear. High in summer, low in fall. Not sure about winter.
Looks like
# some sort of event around the end of 2013, but could just be an end effect.
ggplot(fl,aes(x=sched_dep_time,y=dep_delay)) + geom_point(alpha=0.01) + geom_
smooth()
```

```
# delays increase throughout the day
ggplot(fl,aes(x=distance,y=dep_delay)) + geom_point(alpha=0.01) + geom_smooth
()
```

```
ggplot(fl,aes(x=log(distance),y=dep_delay)) + geom_point(alpha=0.01) + geom_s
mooth()
```

```
# increases with distance -- use log distance
fl <- mutate(fl,logdistance = log(distance)) %>% select(-distance)
ggplot(fl,aes(x=temp,y=dep_delay)) + geom_point(alpha=0.01) + geom_smooth()
```

```
# delays when too hot or too cold
ggplot(fl,aes(x=dewp,y=dep_delay)) + geom_point(alpha=0.01) + geom_smooth()
```

```
# similar to temp
# Etc.
# Replace alt with log(alt)
fl <- mutate(fl,logalt = log(alt)) %>% select(-alt)
```

Split training set in two for tuning

```
set.seed(123)
tr_size <- ceiling(2*nrow(fl)/3)
train <- sample(1:nrow(fl),size=tr_size)
fl_tr <- fl[train,]
fl_te <- fl[-train,]

# baseline to compare learning methods to:
var_dd <- var(fl_te$dep_delay)
var_dd
```

```
## [1] 0.08311941
```

Gam for regression

```
library(gam)
form <- formula(dep_delay ~ s(dep_date) + s(sched_dep_time) + carrier + origi
n + dest + s(logdistance) +
                  s(temp) + s(dewp) + s(humid) + s(wind_dir) + s(wind_speed)
+ precip + s(visib))
gam_fit <- gam(form, data=fl_tr,family=gaussian)
saveRDS(gam_fit,"./gam_fit.rds")

gam_fit<-readRDS("./gam_fit.rds")
summary(gam_fit)

plot(gam_fit,se=TRUE)
```

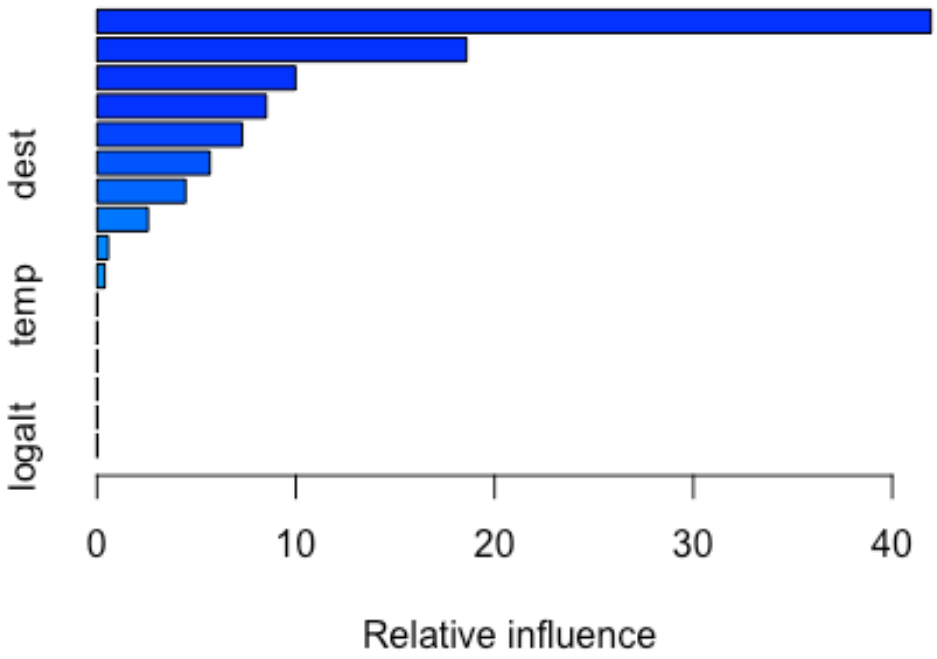
```
gam_pred <- predict(gam_fit,newdata=fl_te)
mse_gam <- mean((fl_te$dep_delay-gam_pred)^2)
mse_gam

## [1] 0.07047458

abs(mse_gam - var_dd)/var_dd

## [1] 0.1521285
```

changing the features format for the regression models

```
library(gbm)
#include this
dep_date_numeric <- as.numeric(fl_tr$dep_date)
dep_date_numeric <- dep_date_numeric - mean(dep_date_numeric)
fl_tr_tem <- mutate(fl_tr,dep_date = dep_date_numeric)
fl_tr_tem <- mutate(fl_tr_tem, origin = factor(origin), dest = factor(dest),
carrier = factor(carrier))

dep_date_numeric <- as.numeric(fl_te$dep_date)
dep_date_numeric <- dep_date_numeric - mean(dep_date_numeric)
fl_te_tem <- mutate(fl_te,dep_date = dep_date_numeric)
fl_te_tem <- mutate(fl_te_tem, origin = factor(origin), dest = factor(dest),
carrier = factor(carrier))
```

boosting

```
gbm_fit <-gbm(dep_delay ~ .,data=fl_tr_tem,distribution="gaussian",
              n.trees = 1000, shrinkage = 0.01)
summary(gbm_fit)
```




```
gbm_pred <- predict(gbm_fit,newdata=fl_te_tem,n.trees = 1000)
mse_gbm <- mean((fl_te_tem$dep_delay-gbm_pred)^2)
mse_gbm
```

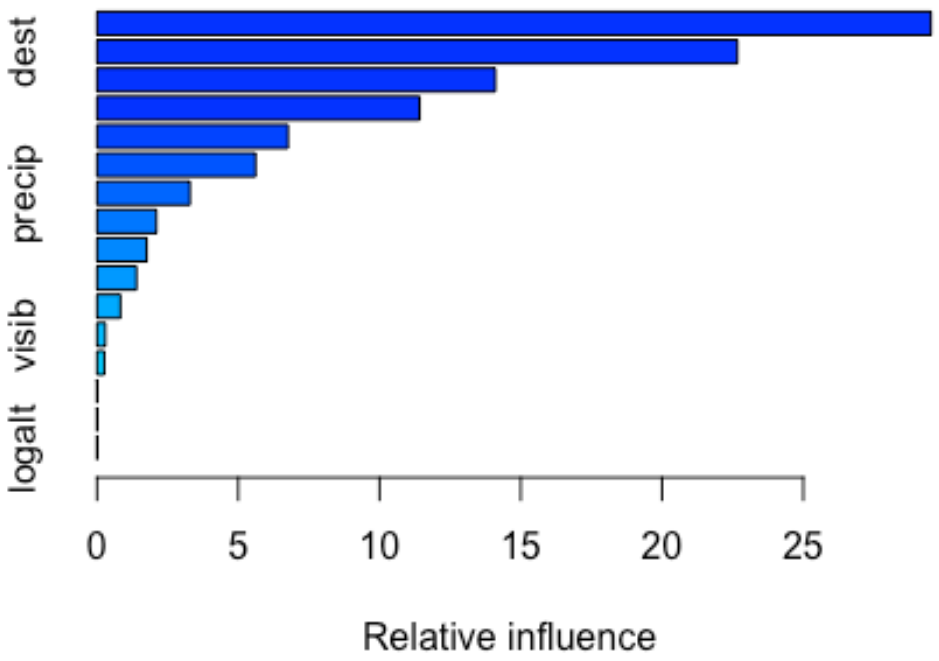
```
## [1] 0.07206566
```

```
abs(mse_gbm - var_dd)/var_dd
```

```
## [1] 0.1329864
```

Boosting Model without shrinkage

```
set.seed(1)
cboost <- gbm(dep_delay ~ ., data=fl_tr_tem, n.trees=1000,distribution = "gau
ssian")
summary(cboost)
```




```
saveRDS(cboost,"./Cboost.rds")
gbm_pred <- predict(cboost,newdata=fl_te_tem,n.trees = 1000)
mse_gbm <- mean((fl_te_tem$dep_delay-gbm_pred)^2)
mse_gbm
```

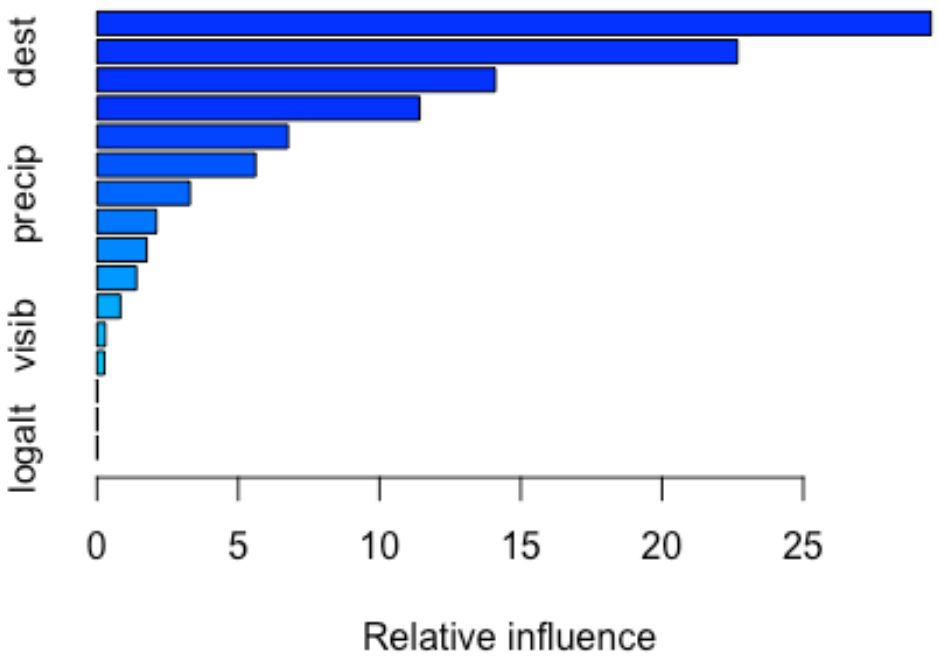
```
## [1] 0.06984851
```

```
abs(mse_gbm - var_dd)/var_dd
```

```
## [1] 0.1596607
```

Random Forest

```
library(randomForest)
randomF <- randomForest(dep_delay ~ .-origin-dest, data=fl_tr_tem, ntree=10,m
try=4)
summary(cboost)
```




```
prediction <- predict(randomF,fl_te_tem)

#mse_gbm <- mean((fl_te_tem$dep_delay-prediction)^2)
saveRDS(randomF,"./randomF.rds")
mse_gbm = mean((fl_te_tem$dep_delay-prediction)^2)
mse_gbm
```

```
## [1] 0.06980387
```

```
abs(mse_gbm - var_dd)/var_dd
```

```
## [1] 0.1601977
```

Lasso

```
library(glmnet)
Xfull <- model.matrix(dep_delay~ ., data=fl_tr_tem)
head(Xfull,n=3)

Y <- fl_tr_tem$dep_delay


lambdas <- 10^{seq(from=-3,to=5,length=100)}
cv.lafit <- cv.glmnet(Xfull,Y,alpha=1,lambda=lambdas)
```

```
la.best.lam <- cv.lafit$lambda.1se
la.best <- glmnet(Xfull,Y,alpha=1,lambda=la.best.lam)
plot(cv.lafit)
```

```
X_te_full <- model.matrix(dep_delay~ ., data=fl_te_tem)
ll <- glmnet(Xfull,Y,alpha=1,lambda=la.best.lam)
saveRDS(ll,"./ll.rds")

gam_fit<-readRDS("./ll.rds")

pred.test=predict(ll,X_te_full)
#mean((fl_te$dep_delay- pred.test)^2)
mse_gbm = mean((fl_te$dep_delay-pred.test)^2)
mse_gbm

## [1] 0.07294183

abs(mse_gbm - var_dd)/var_dd

## [1] 0.1224452

coef(ll)
```

Ridge

```
library(glmnet)
Xfull <- model.matrix(dep_delay~ ., data=fl_tr_tem)
head(Xfull,n=3)

Y <- fl_tr_tem$dep_delay

lambdas <- 10^{seq(from=-3,to=5,length=100)}
cv.lafit <- cv.glmnet(Xfull,Y,alpha=0,lambda=lambdas)
la.best.lam <- cv.lafit$lambda.1se
la.best <- glmnet(Xfull,Y,alpha=0,lambda=la.best.lam)
plot(cv.lafit)
```

```
coef(la.best)

X_te_full <- model.matrix(dep_delay~ ., data=fl_te_tem)
ll2 <- glmnet(Xfull,Y,alpha=1,lambda=la.best.lam)

pred.test=predict(ll2,X_te_full)
#mean((fl_te$dep_delay- pred.test)^2)
```

```
mse_gbm = mean((fl_te$dep_delay-pred.test)^2)
mse_gbm

## [1] 0.08040212

abs(mse_gbm - var_dd)/var_dd

## [1] 0.03269137
```

SVM For Regression (dont run this:takes too long)

```
#tuning
tune.auto <- tune(svm,dep_delay ~ .,data=fl_tr_tem,ranges=list(cost=c(10^{0:1
}),gamma=c(0,0.5,1,2)), kernel="radial")
summary(tune.auto)$performances
#choosing the best model
svm.t <- svm(dep_delay ~ .,data=fl_tr_tem,cost=1,gamma=,0.5 , kernel="radial"
)
summary(svm.t)
s.pred=predict(svm.t,fl_te_tem)
mse_gbm = mean((fl_te_tem$dep_delay-s.pred)^2)
mse_gbm
abs(mse_gbm - var_dd)/var_dd
```

Reading The test Set :

For the test set we have to do all the preprocessing so that it would have the same format as the train set

```
Test_set <- read_csv("fltest.csv.gz")
```

test preprocessing

```
fl2 <- Test_set
for(i in 1:ncol(fl2)) {
  if(typeof(fl2[[i]]) == "character") {
    fl2[[i]] <- factor(fl2[[i]])
  }
}

num_miss <- function(x) { sum(is.na(x)) }
sapply(fl2,num_miss)

fl2 <- fl2%>%
  select(-year.y,-type,-manufacturer,-model,-engines,-seats, -speed, -engine,
-wind_gust,-pressure)
summary(fl2)
```

```r
fl2 <- na.omit(fl2)
summary(fl2)

range(fl2$dep_delay)

fivenum(fl2$dep_delay)

## [1]  -33   -5   -2   11 1137

quantile(fl2$dep_delay,probs = c(0.01,0.05,0.1,0.25,.5,.75,.90,.95,.99))

##  1%  5% 10% 25% 50% 75% 90% 95% 99%
## -12  -9  -7  -5  -2  11  49  88 188

mean(fl2$dep_delay >= 60)

## [1] 0.08249361

fl2%>% arrange(desc(dep_delay)) %>% head(10)

Q3 <- function(x) { quantile(x,probs=.75) }
fl2 %>% group_by(origin) %>%
  summarize(n=n(),med_d = median(dep_delay),Q3_d = Q3(dep_delay), max_d = max
(dep_delay)) %>%
  arrange(desc(Q3_d)) %>% head(10)

## # A tibble: 3 x 5
##   origin     n med_d  Q3_d max_d
##   <fct>  <int> <dbl> <dbl> <dbl>
## 1 EWR    44939    -1    15  1126
## 2 JFK    41468    -1    10  1137
## 3 LGA    39627    -3     6   803

fl2 %>% group_by(month,day) %>%
  summarize(n=n(),med_d = mean(dep_delay),max_d = max(dep_delay)) %>%
  arrange(desc(med_d)) %>% head(10)

## # A tibble: 10 x 5
## # Groups:   month [6]
##    month   day     n med_d max_d
##    <dbl> <dbl> <int> <dbl> <dbl>
##  1     3     8   313  92.1   430
##  2     7     1   322  59.2   272
##  3    12     5   314  53.2   548
##  4     6    30   308  52.6   411
##  5     9     2   309  50.8   514
##  6     6    28   323  49.9   502
##  7     9    12   305  49.8   356
##  8     5    23   275  49.4   475
##  9     6    27   341  49.3   899
## 10     6    13   313  49.3   447
```

```
fl2 %>% mutate(haveprecip = factor(precip>0)) %>% group_by(haveprecip) %>%
  summarize(n=n(),med_d = median(dep_delay),Q3_d = Q3(dep_delay), max_d = max
(dep_delay)) %>%
  arrange(desc(med_d)) %>% head(10)

## # A tibble: 2 x 5
##   haveprecip      n med_d  Q3_d max_d
##   <fct>       <int> <dbl> <dbl> <dbl>
## 1 TRUE         8006     5    42   960
## 2 FALSE      118028    -2     9  1137

den <- nrow(fl2)+1
fl2 <- fl2 %>% mutate(dep_delay = rank(dep_delay)/den)
 ggplot(fl2,aes(x=dep_delay)) + geom_histogram(binwidth=.01)
```

```
library(lubridate)
fl2 <- fl2 %>%
  mutate(dep_date = make_date(year.x,month,day)) %>%
  select(-year.x,-month,-day,-dep_time,-arr_time,-arr_delay,
         -sched_arr_time,-tailnum,-flight,-name,-air_time,
         -hour,-minute,-time_hour,-tz,-dst,-tzone) %>%
  mutate(precip = as.numeric(precip>0))



ggplot(fl2,aes(x=dep_date,y=dep_delay)) + geom_point(alpha=.01) + geom_smooth
()
```

```
# Definitely non-linear. High in summer, low in fall. Not sure about winter.
Looks like
# some sort of event around the end of 2013, but could just be an end effect.
ggplot(fl2,aes(x=sched_dep_time,y=dep_delay)) + geom_point(alpha=0.01) + geom
_smooth()
```

```
# delays increase throughout the day
ggplot(fl2,aes(x=distance,y=dep_delay)) + geom_point(alpha=0.01) + geom_smoot
h()
```

```
ggplot(fl2,aes(x=log(distance),y=dep_delay)) + geom_point(alpha=0.01) + geom_
smooth()
```

```
# increases with distance -- use log distance
fl2 <- mutate(fl2,logdistance = log(distance)) %>% select(-distance)
ggplot(fl2,aes(x=temp,y=dep_delay)) + geom_point(alpha=0.01) + geom_smooth()
```

```
# delays when too hot or too cold
ggplot(fl2,aes(x=dewp,y=dep_delay)) + geom_point(alpha=0.01) + geom_smooth()
```

```
# similar to temp
# Etc.
# Replace alt with log(alt)


fl2 <- mutate(fl2,logalt = log(alt)) %>% select(-alt)
fl2<-fl2[!(fl2$dest=="LEX"),]
dep_date_numeric <- as.numeric(fl2$dep_date)
dep_date_numeric <- dep_date_numeric - mean(dep_date_numeric)
fl3<- mutate(fl2,dep_date = dep_date_numeric)
fl3 <- mutate(fl3, origin = factor(origin), dest = factor(dest), carrier = fa
ctor(carrier))
```

#Gam for test regression

```
gam_pred <- predict(gam_fit,newdata=fl2)
mse_gam <- mean((fl2$dep_delay-gam_pred)^2)
mse_gam
abs(mse_gam - var_dd)/var_dd
```

#boosting for test regression

```
gbm_pred <- predict(gbm_fit,newdata=fl3,n.trees = 1000)
mse_gbm <- mean((fl3$dep_delay-gbm_pred)^2)
mse_gbm

## [1] 0.07177885

abs(mse_gbm - var_dd)/var_dd

## [1] 0.1364369
```

#Random Forest for test

```
prediction <- predict(randomF,fl3)

mse_gbm = mean((fl3$dep_delay-prediction)^2)
mse_gbm

## [1] 0.06933892

abs(mse_gbm - var_dd)/var_dd
```

```
## [1] 0.1657915
```

ridge for test

```
library(glmnet)

X_te_full <- model.matrix(dep_delay~ ., data=fl3)


pred.test=predict(ll2,X_te_full)
#mean((fl_te$dep_delay- pred.test)^2)
mse_gbm = mean((fl3$dep_delay-pred.test)^2)
mse_gbm

## [1] 0.08034946

abs(mse_gbm - var_dd)/var_dd

## [1] 0.0333249
```

Lasso for test

```
X_te_full <- model.matrix(dep_delay~ ., data=fl3)
#ll <- glmnet(Xfull,Y,alpha=1,lambda=la.best.lam)




pred.test=predict(ll,X_te_full)
#mean((fl_te$dep_delay- pred.test)^2)
mse_gbm = mean((fl3$dep_delay-pred.test)^2)
mse_gbm

## [1] 0.07253032

abs(mse_gbm - var_dd)/var_dd

## [1] 0.1273962

coef(ll)
```

Boosting Model without shrinkage on test

```
#Caravan.train <- Caravan[train,]
#Caravan.test <- Caravan[-train,]

gbm_pred <- predict(cboost,newdata=fl3,n.trees = 1000)
mse_gbm <- mean((fl3$dep_delay-gbm_pred)^2)
mse_gbm

## [1] 0.06952649

abs(mse_gbm - var_dd)/var_dd
```

```
## [1] 0.1635348
```

**********Classification**********

preparing data for classification

```
fl_tr_tem_class <- fl_tr_tem %>%
mutate(dep_delay = factor(dep_delay >median(dep_delay )))
fl_te_tem_class <- fl_te_tem %>%
mutate(dep_delay = factor(dep_delay >median(dep_delay )))
#this row had to be deleted in the test set
```

#gam as classifier

```
library(gam)
form <- formula(dep_delay ~ s(dep_date) + s(sched_dep_time) + carrier + origi
n + dest + s(logdistance) +
                  s(temp) + s(dewp) + s(humid) + s(wind_dir) + s(wind_speed)
+ precip + s(visib))
gam_fit2 <- gam(form, family=binomial,data=fl_tr_tem_class)
summary(gam_fit2)

#plot(gam_fit,se=TRUE)
gam.prediction <- (predict(gam_fit2 ,newdata=fl_te_tem_class,type="response")
>0.5)

tt <- table(gam.prediction,fl_te_tem_class$dep_delay)

sum(tt[row(tt)!=col(tt)])/sum(tt)

## [1] 0.3585891

sum(tt)

## [1] 61438
```

Boosting as classifier

```
library(gbm)
set.seed(1)
w.boost <- gbm(I(dep_delay=="True") ~ ., data=fl_tr_tem_class,n.trees=1000,di
stribution="bernoulli",shrinkage = 0.1 )

# default shrinkage = summary(hboost)

#while doing prectition specifiy number of trees, don't need to use all trees
for prediction
prds <- (predict(w.boost,newdata=fl_te_tem_class,n.trees=300,type="response")
>0.5)
```

```
tt1 <- table(prds,fl_te_tem_class$dep_delay)

sum(tt1[row(tt1)!=col(tt1)])/sum(tt1)

## [1] 0.499235

#.19
```

Boosting without shrinkage as classifier

```
library(gbm)
set.seed(1)
w.boost <- gbm(I(dep_delay=="True") ~ ., data=fl_tr_tem_class,n.trees=1000,di
stribution="bernoulli" )

# default shrinkage = summary(hboost)

#while doing prectition specifiy number of trees, don't need to use all trees
for prediction
prds <- (predict(w.boost,newdata=fl_te_tem_class,n.trees=300,type="response")
>0.5)


tt1 <- table(prds,fl_te_tem_class$dep_delay)

sum(tt1[row(tt1)!=col(tt1)])/sum(tt1)

## [1] 0.499235

#.19
```

Naive Bayes as classifier

```
library(e1071)
set.seed(123)
nvb <- naiveBayes(dep_delay~., data=fl_tr_tem_class)
nvbpred <- predict(nvb,fl_te_tem_class)
#mean(fl_te_tem$dep_delay == nvb.pred)
tt2 <- table(nvbpred,fl_te_tem$dep_delay)
sum(tt2[row(tt2)!=col(tt2)])/sum(tt2)

## [1] 0.9999837
```

RandomForest as classifier

```
library(randomForest)
set.seed(123)
randomF2<- randomForest(dep_delay~.-origin-dest,data=fl_tr_tem_class, ntree=1
0,mtry=4)
rand.prediction <- predict(randomF2,newdata=fl_te_tem_class, type="class")
```

```
tt3 <- table(rand.prediction,fl_te_tem_class$dep_delay)
sum(tt3[row(tt3)!=col(tt3)])/sum(tt3)

## [1] 0.3693805
```

0.3693805

SVM as classifier kernel radial

```
library(e1071)
svm.heart <- svm(dep_delay~.,type="C-classification",cost=1,data=fl_tr_tem,kernel="radial",gamma=1/2)
pp <- predict(svm.heart,newdata=fl_te_tem)
saveRDS(pp,"./pp.rds")
pp<-readRDS("./pp.rds")
tt5<- table(pp,fl_te_tem$dep_delay)
sum(tt5[row(tt5)!=col(tt5)])/sum(tt5)
```

SVM as classifier with kernel linear

```
library(e1071)
svm.heart <- svm(dep_delay~.,type="C-classification",cost=1,data=fl_tr_tem,kernel="linear",gamma=1/2)
saveRDS(pp,"./linearmodel.rds")
pp <- predict(svm.heart,newdata=fl_te_tem)
tt5<- table(pp,fl_te_tem$dep_delay)
sum(tt5[row(tt5)!=col(tt5)])/sum(tt5)
```

Logistic Regression

```
log_fit <- glm(dep_delay~.,data=fl_tr_tem_class, family=binomial())
predDirection = (predict(log_fit,fl_te_tem_class,type="response")>0.5)
tt5 <- table(predDirection,fl_te_tem_class$dep_delay)

sum(tt5[row(tt5)!=col(tt5)])/sum(tt5)

## [1] 0.3723103
```

LDA for classification

```
library(MASS)
lda_fit <- lda(dep_delay~.,data=fl_tr_tem_class, family=binomial())
e.preds <-predict(lda_fit,fl_te_tem_class,type="response")
mean(e.preds$class== fl_te_tem_class$dep_delay)

## [1] 0.6279339

table(e.preds$class,fl_te_tem_class$dep_delay)
```

```
##
##         FALSE  TRUE
##   FALSE 19883 11976
##   TRUE  10883 18696
```

preparing data for classification test

```
fl_tr_tem_class_test <- fl3  %>%
mutate(dep_delay = factor(dep_delay >median(dep_delay )))
fl_te_tem_class_test <- fl3  %>%
mutate(dep_delay = factor(dep_delay >median(dep_delay )))
fl_te_tem_class_test<-fl_te_tem_class_test[!(fl_te_tem_class_test$dest=="LEX"
),]
```

SVM as classifier kernel radial testing

```
pp <- predict(svm.heart,newdata=fl_tr_tem_class_test)
tt5<- table(pp,fl_te_tem$dep_delay)
sum(tt5[row(tt5)!=col(tt5)])/sum(tt5)
```

SVM as classifier kernel linear testing

```
pp <- predict(svm.heart,newdata=fl_tr_tem_class_test)
tt5<- table(pp,fl_te_tem$dep_delay)
sum(tt5[row(tt5)!=col(tt5)])/sum(tt5)
```

LDA for testing

```
library(MASS)
e.preds <-predict(lda_fit,fl_te_tem_class_test,type="response")
mean(e.preds$class== fl_te_tem_class_test$dep_delay)

## [1] 0.6287322

table(e.preds$class,fl_te_tem_class_test$dep_delay)

##
##         FALSE  TRUE
##   FALSE 41198 24455
##   TRUE  22337 38043
```

#Logistic Regression

```
predDirection = (predict(log_fit,fl_te_tem_class_test,type="response")>0.5)
tt5 <- table(predDirection,fl_te_tem_class_test$dep_delay)

sum(tt5[row(tt5)!=col(tt5)])/sum(tt5)

## [1] 0.3714265
```

```
library(randomForest)

rand.prediction <- predict(randomF2,newdata=fl_te_tem_class_test, type="class
")
tt3 <- table(rand.prediction,fl_te_tem_class_test$dep_delay)
sum(tt3[row(tt3)!=col(tt3)])/sum(tt3)

## [1] 0.3676418
```

Naive Bayes as classifier

```
nvbpred <- predict(nvb,fl_te_tem_class_test)
#mean(fl_te_tem$dep_delay == nvb.pred)
tt2 <- table(nvbpred,fl_te_tem_class_test$dep_delay)
sum(tt2[row(tt2)!=col(tt2)])/sum(tt2)

## [1] 0.4015298
```

Boosting without shrinkage as classifier

```
library(gbm)
set.seed(1)

prds <- (predict(w.boost,newdata=fl_te_tem_class_test,n.trees=300,type="respo
nse")>0.5)


tt1 <- table(prds,fl_te_tem_class_test$dep_delay)

sum(tt1[row(tt1)!=col(tt1)])/sum(tt1)

## [1] 0.495886

#.19
```

GAM as classifier

```
gam.prediction <- (predict(gam_fit2 ,newdata=fl_te_tem_class_test,type="respo
nse")>0.5)

tt <- table(gam.prediction,fl_te_tem_class_test$dep_delay)

sum(tt[row(tt)!=col(tt)])/sum(tt)

## [1] 0.3584458

sum(tt)

## [1] 126033
```