Titanic: Machine Learning from Disaster

Casey Quinn

February 10, 2016

```
library(dplyr) # Data wrangling
library(tidyr) # Data wrangling
library(caret) # Machine learning
library(ggplot2) # Plotting
library(VIM) # Data visulazation
library(mice) # Data visulazation
library(stringr) # For string manipulation
library(xtable)
```

Overview

The Titanic: Machine Learning from Disaster competition posted on Kaggle was used to learn about and apply various machine learning techniques including K-nearest neighbors (KNN), naive Bayes, logistic regression, classification trees, bagging, random forests, boosting, and support vector machines (SVM). This document will provide an overview of the techniques applied and a comparison of the results.

Data

The Kaggle website provides two primary data sets; train.csv and test.csv. The train dataset consists of 891 observations and 11 potential predictor variables (PassengerId, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked) and 1 outcome variable (Survived). The test dataset consists of 418 observations and the same 11 predictor variables and no oucome varible.

The data files were first downloaded and loaded into R and a couple of the variables were reclassified:

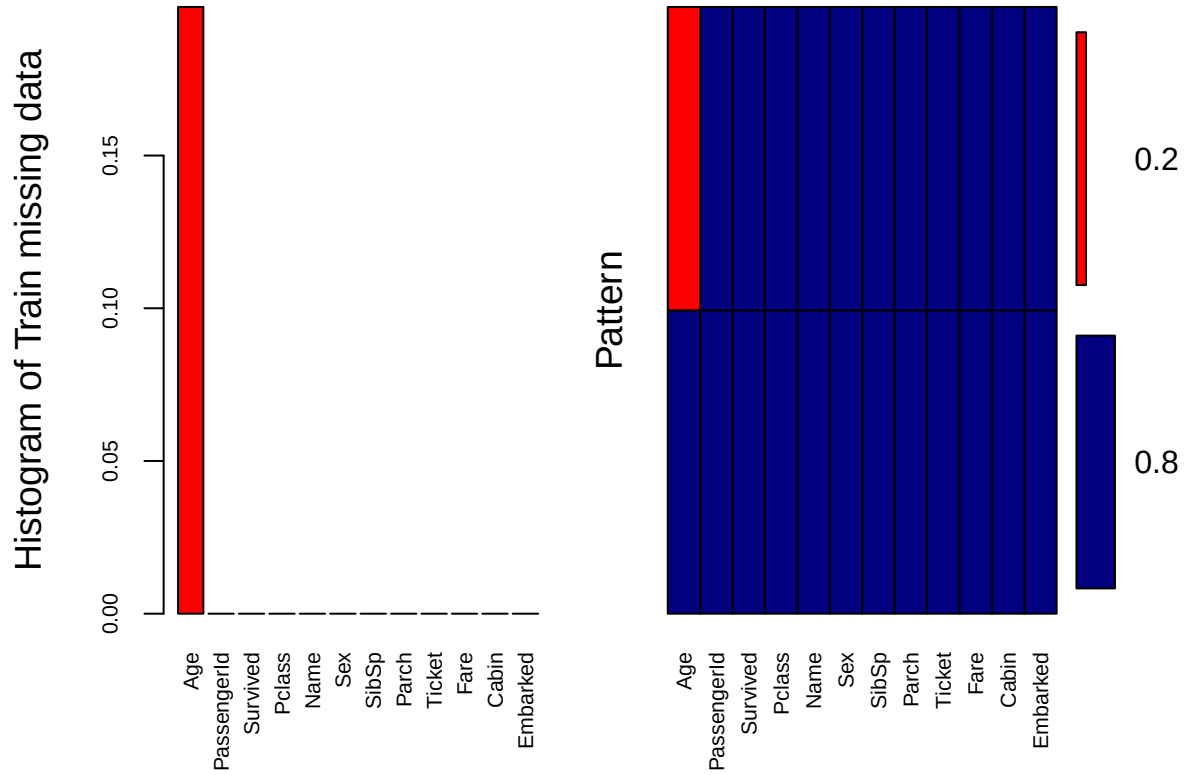
```
my_dir <- "C:/Docs/Dropbox/PhD/classes/STAT695/Titanic/"
setwd(my_dir)

train <- read.csv("data/train.csv")%>%
  mutate(Survived = factor(Survived),
  Pclass = factor(Pclass),
  Name = as.character(Name),
  Sex = factor(Sex))

test <- read.csv("data/test.csv")%>%
  mutate(Pclass = factor(Pclass),
  Name = as.character(Name),
  Sex = factor(Sex))
```

Once loaded into R, the datasets were parsed for missing values using the mice and VIM packages. The results for both the train and test datasets are shown in the tables and figures below.

```
aggr_plot <- aggr(train, col=c('navyblue','red'), numbers=TRUE, sortVars=TRUE,
                  labels=names(data), cex.axis=.7, gap=3,
                  ylab=c("Histogram of Train missing data","Pattern"))
```




```
##
##  Variables sorted by number of missings:
##      Variable     Count
##           Age 0.1986532
##   PassengerId 0.0000000
##      Survived 0.0000000
##        Pclass 0.0000000
##          Name 0.0000000
##           Sex 0.0000000
##         SibSp 0.0000000
##         Parch 0.0000000
##        Ticket 0.0000000
##          Fare 0.0000000
##         Cabin 0.0000000
##      Embarked 0.0000000
```

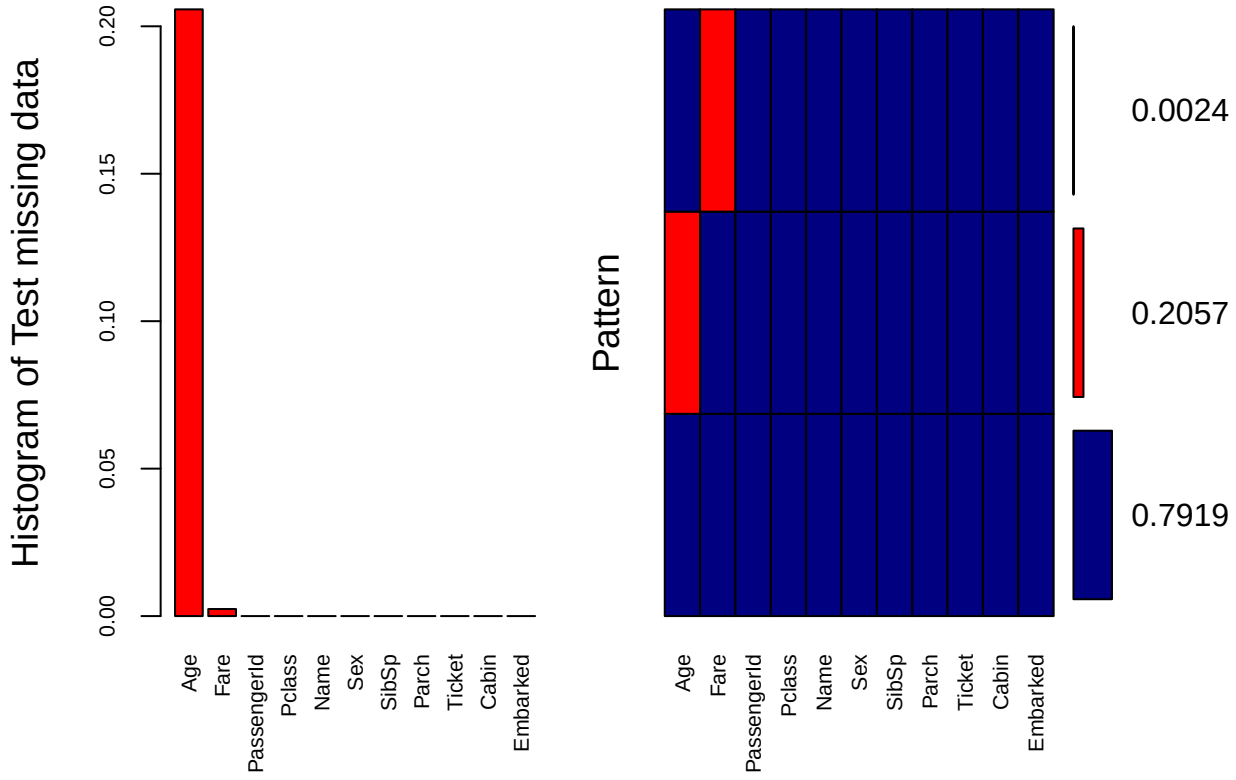
```
train_md <- md.pattern(train)
```

```
## Warning in data.matrix(x): NAs introduced by coercion
```

```
knitr::kable(train_md)
```

	PassengerId	Survived	Pclass	Sex	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age	Name	
714	1	1	1	1	1	1	1	1	1	1	1	0	1
177	1	1	1	1	1	1	1	1	1	1	0	0	2
	0	0	0	0	0	0	0	0	0	0	177	891	1068

```
aggr_plot <- aggr(test, col=c('navyblue','red'), numbers=TRUE, sortVars=TRUE,
                  labels=names(data), cex.axis=.7, gap=3,
                  ylab=c("Histogram of Test missing data","Pattern"))
```




```
##
##  Variables sorted by number of missings:
##     Variable       Count
##          Age 0.205741627
##         Fare 0.002392344
##  PassengerId 0.000000000
##       Pclass 0.000000000
##         Name 0.000000000
##          Sex 0.000000000
##        SibSp 0.000000000
##        Parch 0.000000000
##       Ticket 0.000000000
```

```
##         Cabin 0.000000000
##      Embarked 0.000000000
```

```
test_md <- md.pattern(test)
```

```
## Warning in data.matrix(x): NAs introduced by coercion
```

```
knitr::kable(test_md)
```

	PassengerId	Pclass	Sex	SibSp	Parch	Ticket	Cabin	Embarked	Fare	Age	Name	
331	1	1	1	1	1	1	1	1	1	1	0	1
86	1	1	1	1	1	1	1	1	1	0	0	2
1	1	1	1	1	1	1	1	1	0	1	0	2
	0	0	0	0	0	0	0	0	1	86	418	505

Methods

In this section snipets of my code for each of the models will be displayed for reference.

KNN

```
train_x <- select(train, Fare)
train_y <- train$Survived
test_x <- select(test,Fare)
test_x$Fare[NAindex_test_fare] <- mean(test$Fare, na.rm = TRUE)
set.seed(1201)
train_pred_fare <- knn(train_x, train_x, train_y, k = 1)
```

Trees

```
my_tree_two <- rpart(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,
                     data = train, method ="class")
pred_train <- predict(my_tree_two, train, type = "class")

##################
tree.Titanic =tree(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,  train)
summary(tree.Titanic)
#creates a regression tree for some reason, thus used the probability trick from
#logistic regression...
tree.pred=predict (tree.Titanic, test)
pred_test=rep (0 , length(tree.pred))
pred_test[tree.pred >.5]=1
```

Random Forest / Bagging

```
bag.my_forest <- randomForest(as.factor(Survived) ~ Pclass + Sex + Age + SibSp + Parch +
                                Fare + Embarked + Title, data = train,
                              importance = TRUE, ntree = 1000, mtry = 8)
pred_train <- predict(bag.my_forest, train)

##################
my_forest <- randomForest(as.factor(Survived) ~ Pclass + Sex + Age + SibSp + Parch +
                            Fare + Embarked + Title, data = train,
                          importance = TRUE, ntree = 1000)
pred_train <- predict(my_forest, train)
```

Logistic Regression

```
glm.fit=glm(Survived~Sex+Fare , data=train ,family =binomial )
glm.probs = predict(glm.fit ,type ="response")
glm.pred=rep (0 , length(glm.probs))
glm.pred[glm.probs >.5]=1
```

LDA

```
lda.fit=lda(Survived~Sex+Fare ,data=train)
lda.pred = predict(lda.fit)
```

Naive Bayes

```
nb_mod <- naiveBayes(Survived ~ Pclass + Sex + Age + Fare, data = train)
pred_train <- predict(nb_mod, train)
```

SVM

```
set.seed(1)
svmfit =svm(Survived ~ Pclass + Sex + SibSp + Parch, data=train,
            kernel ="polynomial", degree =2,  cost =1)
svm.predtrain=predict(svmfit, train)

#Tuning of the SVM with e1071
tune.out=tune(svm, Survived ~ Pclass + Sex + SibSp + Parch, data=train,
              kernel ="polynomial",
              ranges =list(cost=c(0.1, 1, 10, 100), degree=c(3, 4) ))
pred.besttrain=predict(tune.out$best.model, train)
```

Results

The results from simulations ran for a variety of different models are shown in the table below. As one can see most of the models performed fairly equally, and one should also note little time was spent trying to optimize any of the models. The Traiing Accuracy was determined used code similar to this:

```
sum(pred_train == train$Survived) / length(pred_train)
```

Model	Package/function	Description	Training Accuracy	Test Accuracy
KNN	e1071/knn	Fare K=5	0.776	0.660
KNN	e1071/knn	Sex&Fare K = 5	0.767	0.766
Tree	rpart	All Variables + Derived Values	0.840	0.804
Tree	rpart	All Varibles	0.840	0.785
Tree	tree	All	0.616	0.780
Random Forest	rPart/randomForest	All	0.912	0.799
Bagging	rPart/randomForest	All	0.984	0.737
Logistic Regresion	glm	Sex&Fare	0.764	0.761
LDA	MASS/lda	Sex&Fare	0.765	0.761
Naive Bayes	e1071/naiveBayes	Pclass	0.679	0.655
Naive Bayes	e1071/naiveBayes	Pclass + Sex + Age + Fare	0.777	0.727
SVM	e1071/svm poly2 c=1	Pclass Sex Sib Par	0.810	0.770
SVM	e1071/svm tuned	Pclass Sex Sib Par	0.807	0.775
SVM	e1071/svm gamma = 0.5 c=1	Pclass Sex Sib Par	0.816	0.804

Conclusions

While there are many differnt machine learning techniques, most of them had similar performance for estimating the survivial of the Titanic passengers with ~70-80% accuracy. Hopefully this is primarily driven by the survival outcome being somewhat random rather than demonstrating a deficiency in the machine learning techniques. The individual packages seemed to be quite fincky and required different data cleaning techniques, whilst the `caret package` provides the ability to run a large variety of different models as well as the ability to tune the different parameters for each model using the ***train*** function. The goal will be to learn how to leverage the caret package more to reduce the amount of necessary data cleaning. Additionally future focus on how to deal with NA values in both the training and test data also needs to be done. There are imputation packages such as mice and VIM that seem to have good potential for helping fill in the missing values if there are few or the avoidance of certain data if too much data is missing!