DV Final Project

Dan Vachalek

12/14/2019

Final Project

#Problem statement

##VC investors often find that they need to aggressively evaluate all opportunities that come across their desk, because one correct investment decision can offest the total losses of their portfolio. Time, as we all know, is limited. Balancing limited time with the fear of missing out on the next "big thing" presents an opportunity for us to leverage machine learning. It would be incredibly helpful for VC firms to have a model that helps give a simple binary result of whether a startup that comes across their desk is worth further due diligence or worth passing on. The goal of this project is to see if we can build a model for identifying potential $10M plus startups based on the attributes available from angel.co, a well known site for startups. We will attempt to build a classification model that outputs a "yes" or "no" indicator for whether the company is worth a deep dive analysis from a financial analyst that works for the VC firm. (A very expensive activity)

#Data Description ## This data was sourced from angel.co and was downloaded in pieces with various filters applied, such as "Had a founder that worked previously at Google. It was then rbinded together for analysis. Angel.co is a well known site where employees look for jobs at startups, so the incentive to post accurate and timely data on the company as a tool for recruiting new talent is very high.

##This data has 18 variables #Name, Factor w/ 744 levels (The name of the company) #Profile.URL, Factor w/ 744 levels (Their profile on the site) #Signal, int 1-5 (a score of 1-5, that is supposed to measure startup quality) #Joined, Factor w/ 88 levels (month and year that the company created their profile on the site) #Location, Factor w/ 109 (Self-identified location where the startup operates) #State, Factor with 29 levels (state in which startup operates) #Region, Factor with 5 levels (US region in which startup operates) #east_coast, Factor with 2 levels (yes or no, whether the company operates on the east coast) #west_coast, Factor with 2 levels (yes or no, whether the company operates on the west coast) #Market, Factor with 284 levels (Self-identified vertical/market that the startup competes in) #Website, Factor with 744 levels (The startups business website) #Employees, Factor with 8 levels (Ranges for the number of employees at the startup) #Stage, Factor with 11 levels (The most recent stage of funding that the startup has secured) #Total.raised Factor with 484 levels, (The total amount of capital in USD that the startup has raised) #filter_labelapple_alumni, Factor with 2 levels (Whether the company has a founder that worked previously at Apple) #filter_labelgoogle_alumni, Factor with 2 levels (Whether the company has a founder that worked previously at Google) #filter_labelharvard_alumni, Factor with 2 levels (Whether the company has a founder that graduated from Harvard) #filter_lavelyale_alumni, Factor with 2 levels (Whether the company has a foundre that graduated from Yale)

Data clean-up/prep & EDA

#The data could originally only be downloaded in CSV's with 100 records each. So I download many versions with various filters applied, and then rbinded the dataframes together, and merged distinct records. The example of the workflow is below

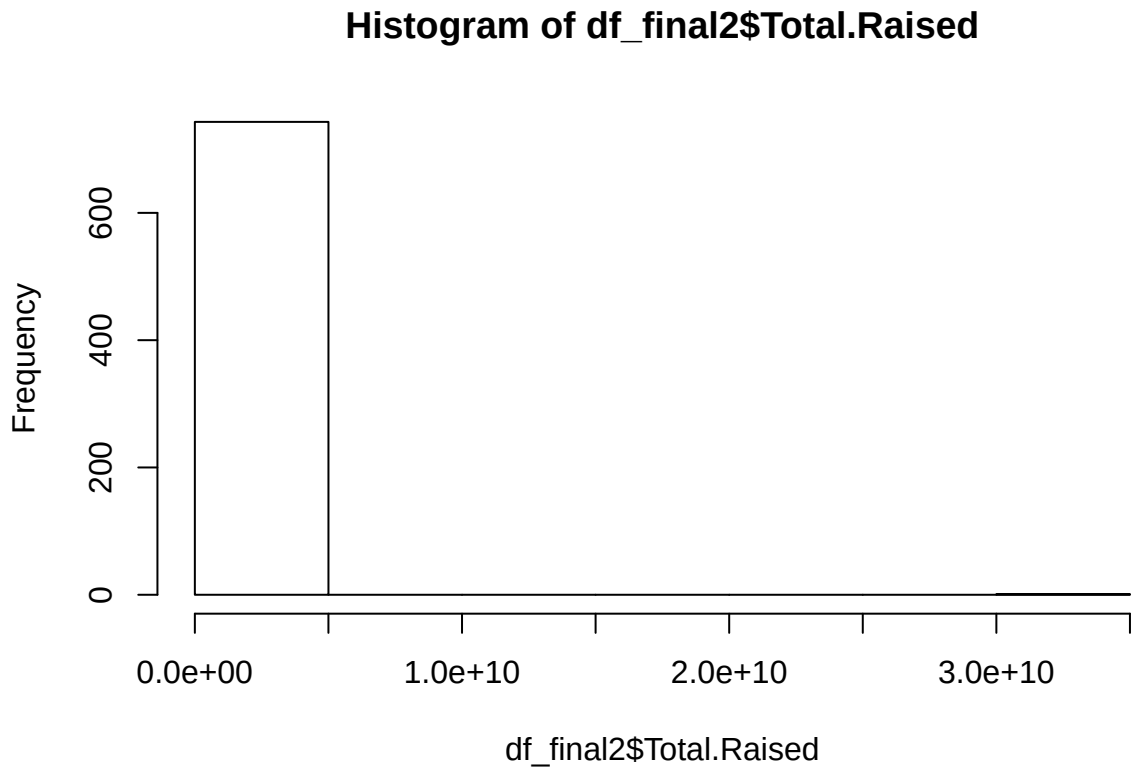
```
# Example pre-work to read in the CSV's from angel.co (The
# site unfortunately recently closed their API, which makes
# getting data very manual, consolidated the approach for
# readability)

# Read in all the CSV's as dataframes, examples below df1 <-
# read.csv('C:/Users/Dv/Downloads/companies (1).csv' , header
```

```
# = TRUE, sep = ',') df2 <-
# read.csv('C:/Users/Dv/Downloads/comp3.csv' , header = TRUE,
# sep = ',') df6 <-
# read.csv('C:/Users/Dv/Downloads/comp6_yale.csv' , header =
# TRUE, sep = ',') etc.

# Bind together the Df's and deduplicate for to get distinct
# companies (some of the CSV's pulled overlapping records)
# df_apple <- rbind(df_apple,df_apple2,df_apple3,df_apple4)
# df_apple_distinct <- distinct(df_apple) df_google_distinct
# <- distinct(df_google) etc.

# df_yale <- rbind(df1,df2,df3,df4,df5) bind together all
# Yale CSVs df_yale_distinct <- distinct(df_yale) deduplicate
# companies that were fetched manually etc.
```

#Here is the final CSV that I ended up reading back into R

```
df_final <- read.csv("C:/Users/Danny Vachalek/Downloads/dummy1 - dummy2.csv",
    header = TRUE, sep = ",")

# Checking that all variables types are correct...
str(df_final)

# Continuing the Data Prep Let's add a column for our target
# prediction. Since we want to predict whether a startup will
# be valued at more than $25M, let's bin our funding column
# into two factors - Secured more than $5M in funding or not.
# Typically startups fetch ~5-8x average valuation multiples,
# so ~$5M in funding means the startup is valued at ~$25-40M.

# Total.Raised was read in as a factor instead of numeric.
str(df_final$Total.Raised)

# The commas seem to be the issue. Let's remove them and
# convert to numeric.
df_final$Total.Raised <- gsub(",", "", df_final$Total.Raised)  #remove commas
df_final$Total.Raised <- as.numeric(as.character(df_final$Total.Raised))  #convert to character, then t
str(df_final$Total.Raised)  #recheck

# Change total.raised to a binary yes/no character, and start
# a new dataframe to preserve our progress.
df_final2 <- df_final %>% mutate(`25M+` = case_when(Total.Raised >=
    5e+06 ~ "yes", Total.Raised < 5e+06 ~ "no"))

str(df_final2$`25M+`)  #recheck
# Great, now we have the character needed for $25M+ value,
# but it really needs to be a factor
df_final2$`25M+` <- as.factor(df_final2$`25M+`)  #convert to factor
str(df_final2$`25M+`)  #recheck

# Recheck structure
str(df_final2)
```

```r
hist(df_final2$Total.Raised)  #The total value raised is extremely skewed, and this highlights why it's
```




```r
# Signal may not be useful, but let's change it to a factor
# just in case.
df_final2$Signal <- factor(df_final2$Signal, levels = c("1",
    "2", "3", "4", "5"))
str(df_final2$Signal)  #recheck structure

# We likely won't need the below columns, so removing them.

df_final2 <- subset(df_final2, select = -Joined)  #Multicollinearity issue, as the stageup age correlat
df_final2 <- subset(df_final2, select = -Website)  #No logical connection to funding
df_final2 <- subset(df_final2, select = -Profile.URL)  #No logical connection to funding
df_final2 <- subset(df_final2, select = -Name)  #No logical connection to funding
df_final2 <- subset(df_final2, select = -Market)  #Too many factor levels to be useful for now
df_final2 <- subset(df_final2, select = -Location)  #Too many factors levels to be useful for now
df_final2 <- subset(df_final2, select = -Stage)  #Multicollinearity issue, and the stage is set after t
df_final2 <- subset(df_final2, select = -Employees)  #Multicollinearity issue with value, as employees

# Now we need to remove total raised from df_final2 as it
# will likely be almost a perfect predictor for our yes/no
# flag of 100M+ and our model will suffer from
# multicollinearity.
df_final2 <- subset(df_final2, select = -Total.Raised)
str(df_final2)  #recheck
```

Now that we have our data organized, let's check out the summary

```
summary(df_final2)  #We have a lot of California and New York-based startups, most of the companies ope
```

#Modeling #The algorithms we are going to use to predict the 25M+ flag are logistic regression, neural net and extreme gradient boosting.

#First we will prepare test and training sets that we'll use for the models

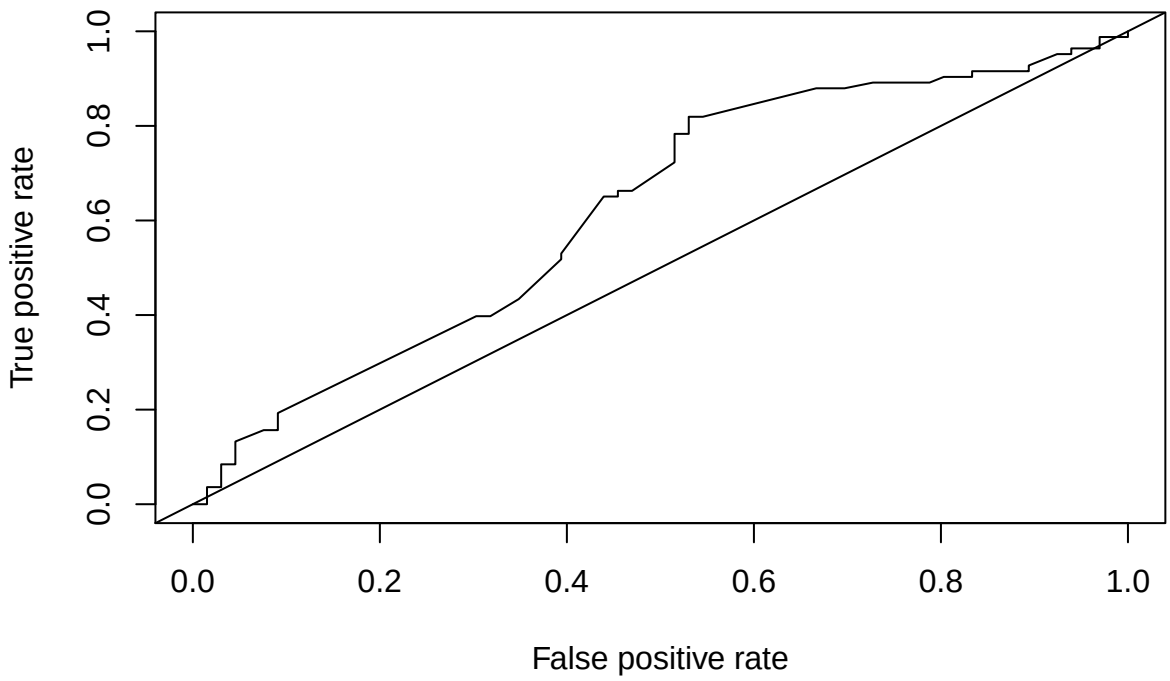
```
# Are there any missing values?
apply(df_final2, 2, function(x) any(is.na(x)))  ## False, we can proceed

features2 = df_final2[, -10]  # obtain the features, all columns except the last one. The target variab


X2 = dummy.data.frame(features2)  # One hot encode the features
dim(X2)  #give the dimensions of the df. We can see that we now have the one-hot encoded version of our
head(X2)  #show the first few records just to doublecheck


target2 = df_final2[, 10]  # obtain the target variable
head(target2)  #doublecheck that we have the target variable in target2

# Transform the target2 variable to 0 and 1
Y2 = dummy.data.frame(as.data.frame(target2))  #one hot encode
print(Y2)  #Check new DF with one-hot encoding
Y2 = Y2[, 2]  #select the 2nd col as the intended predictor
data2 = cbind(X2, Y2)  #combine together with X2
head(data2)  #check progress, we have the Y2 column with 0 and 1 appropriately binded to X2

Data2 = data.matrix(data2)  # convert data into numeric matrix

# Split the data into training & testing sets, 80:20
p2 = ncol(Data2)  #get # of col and save as variable p2
N2 = nrow(Data2)  #get # of rows and save as variable N2
Ind2 = sample(N2, N2 * 0.8, replace = FALSE)  #take a sample without replacement, save as Ind2 variable
Ind2[1:100]  #check values in Ind2

# Create test and training sets that don't overlap below

# Create training set
Y_train2 = as.factor(Data2[Ind2, p2])
X_train2 = as.data.frame(Data2[Ind2, -p2])

# Create testing set
Y_test2 = as.factor(Data2[-Ind2, p2])
X_test2 = as.data.frame(Data2[-Ind2, -p2])
# Double check the dimensions of the training and test sets X
# and Y to make sure we did this correctly
dim(X_train2)  #595 x 50
length(Y_train2)  #595

dim(X_test2)  #149
length(Y_test2)  #149
# Looks good!
```

#Lets run the train data through our logistic regression and store the model

```
fm_logit2 = glm(Y_train2 ~ ., data = X_train2, family = binomial)  #predict the Y_train2 response varia
summary(fm_logit2)
# Looks like Googleno, Harvardno and Yaleno are significant
```

Fit the model

```
logit.preds = predict(fm_logit2, newdata = as.data.frame(X_test2),
    type = "response")
logit.pred = prediction(logit.preds, Y_test2)
logit.perf = performance(logit.pred, "tpr", "fpr")
plot(logit.perf)
abline(a = 0, b = 1)
```




```
# ROC curve looks okay, considering this is a real life use
# case and not data typically used for a machine learning
# class One of the most important metrics for a classifcation
# model is the area under the ROC curve, which essentially
# tells us the probability of being able to distinguish
# between positive and negative outputs.

logit.auc = performance(logit.pred, "auc")
# Area under curve (One of the most important metrics of a
# classification model)
reg.area = logit.auc@y.values[[1]]
print(reg.area)
```

```
# 0.651 is still better than random chance. I think any VC
# firm would be satisfied with an improvement over random
# chance, because this saves lots of analyst time.

# Below makes the confusion matrix and prints out the results
Predicted = logit.preds >= 0.5
Actual = Y_test2

# Store in table for the matrix
tabs.reg <- table(Predicted, Actual)
dimnames(tabs.reg)[[1]] <- c(levels(Y_train2)[1], levels(Y_train2)[2])
# print the confusion matrix
print(confusionMatrix(tabs.reg, positive = "1"), mode = "everything")

# Confusion matrix looks OK- There are a lot of actual 1's
# that were predicted as zeros We are decently successfully
# saying pass on this deal, without throwing out a ton of
# valid deals. However, we are still suggesting to the VC
# firm that 43 companies are worth investigating in, when in
# reality they weren't worth the time. #F1 : 0.6919 is decent
# for a real life scenario like this.  Actual Predicted 0 1 0
# 28 14 1 43 64
```

Let's tune the model using elasticnet

```
# set training control
train_control = trainControl(method = "cv", number = 5, search = "random",
    verboseIter = FALSE, savePredictions = TRUE)

# train the model using elasticnet
lambda = 10^seq(10, -2, length = 100)
alpha = runif(100, 0, 1)  # 20 random values between 0, 1
elasticNet_logit = caret::train(Y_train2 ~ ., data = cbind(Y_train2,
    X_train2), method = "glmnet", family = "binomial", trControl = train_control,
    metric = "Accuracy", tuneGrid = as.data.frame(cbind(lambda,
        alpha)))
```

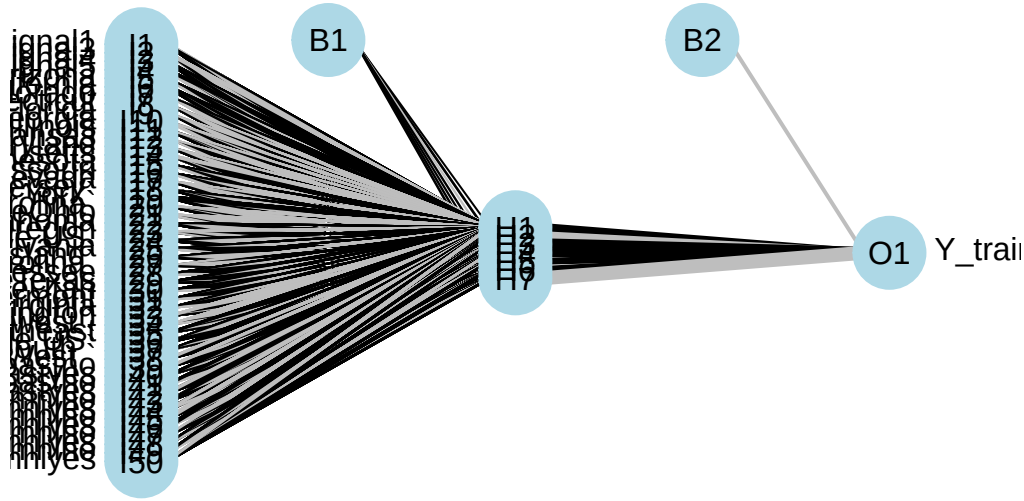
Make new predictions with our tuned Elastic net

```
logit.preds2 = predict(elasticNet_logit, newdata = as.data.frame(X_test2),
    type = "prob")
logit.pred2 = prediction(logit.preds2[, 2], Y_test2)
logit.perf2 = performance(logit.pred2, "tpr", "fpr")
logit.auc2 = performance(logit.pred2, "auc")
# Check Area under curve again
reg.area2 = logit.auc2@y.values[[1]]
print(reg.area2)
# 0.682 is better than our previously un-tuned model! We
# gained a 3% point boost.

# Build and print the confusion matrix again
Predicted2 = logit.preds2[, 2] >= 0.5
```

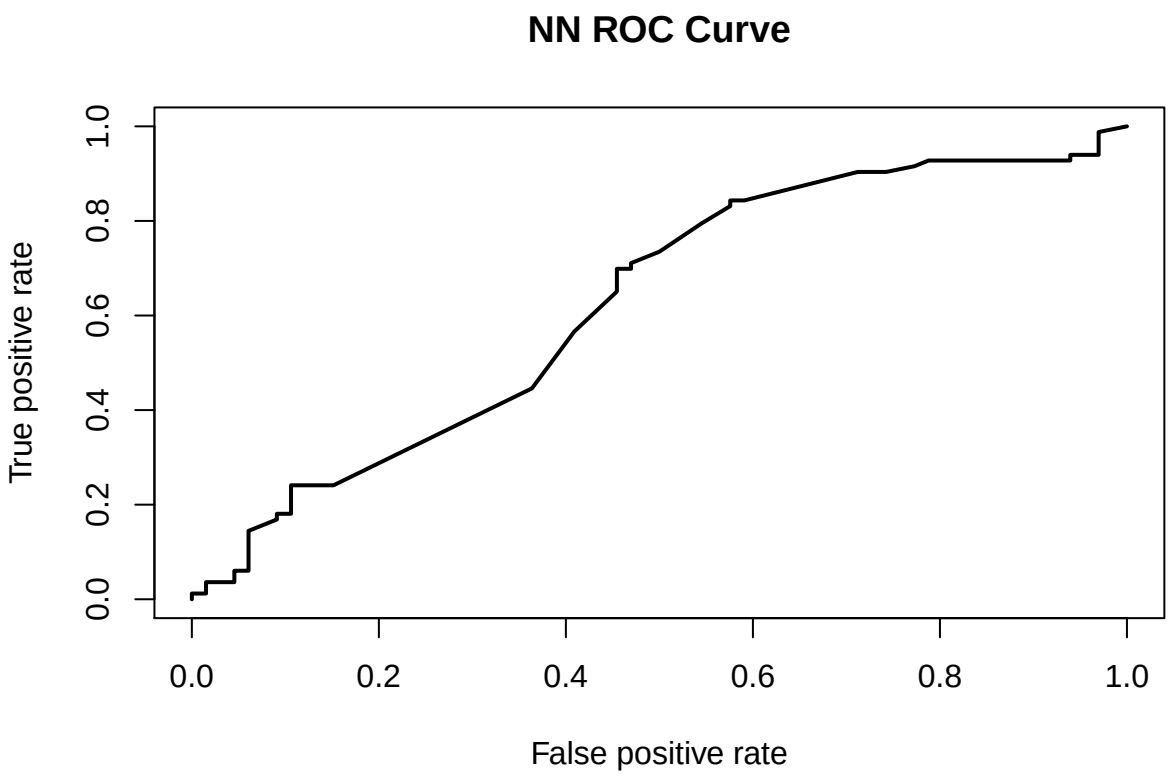
```r
Actual2 = Y_test2
tabs.reg2 <- table(Predicted2, Actual2)
dimnames(tabs.reg2)[[1]] <- c(levels(Y_train2)[1], levels(Y_train2)[2])
print(confusionMatrix(tabs.reg2, positive = "1"), mode = "everything")

# Not bad, we gained another correctly classified company,
# moving 1 company from a false positive, to a true negative.
# Overall we gained a minor boost in F1, which went from
# 0.6919 to 0.6957.  Actual2 Predicted2 0 1 0 29 14 1 42 64
```

Round 2 - Let's try Neural Net

```r
# Fit the NN model with the same training and test dataset
nnet.fit = nnet(Y_train2 ~ ., data = X_train2, size = 7, maxit = 10000,
    decay = 1e-04)
nnet.preds = predict(nnet.fit, newdata = X_test2, type = "raw")
nnet.pred = prediction(nnet.preds, Y_test2)
nnet.perf = performance(nnet.pred, "tpr", "fpr")
nnet.auc = performance(nnet.pred, "auc")

# Check and print the AUC
nnet.area = nnet.auc@y.values[[1]]
print(nnet.area)  #0.6254 Initially not as good as the tuned logistic model

# Create and print the confusion matrix
Predicted = nnet.preds >= 0.5
Actual = Y_test2
tabs.nnet <- table(Predicted, Actual)
dimnames(tabs.nnet)[[1]] <- c(levels(Y_train2)[1], levels(Y_train2)[2])
print(confusionMatrix(tabs.nnet, positive = "1"), mode = "everything")
# The results are a minor improvement over the tuned elastic
# net, so perhaps we'll get and even better result after
# tuning the Neural network Actual Predicted 0 1 0 33 18 1 38
# 60 Better than random chance, but not groundbreaking
library(NeuralNetTools)
plotnet(nnet.fit)  #What does the Neural net look like
```




```
plot(nnet.perf, col = 1, lwd = 2, main = "NN ROC Curve")
```

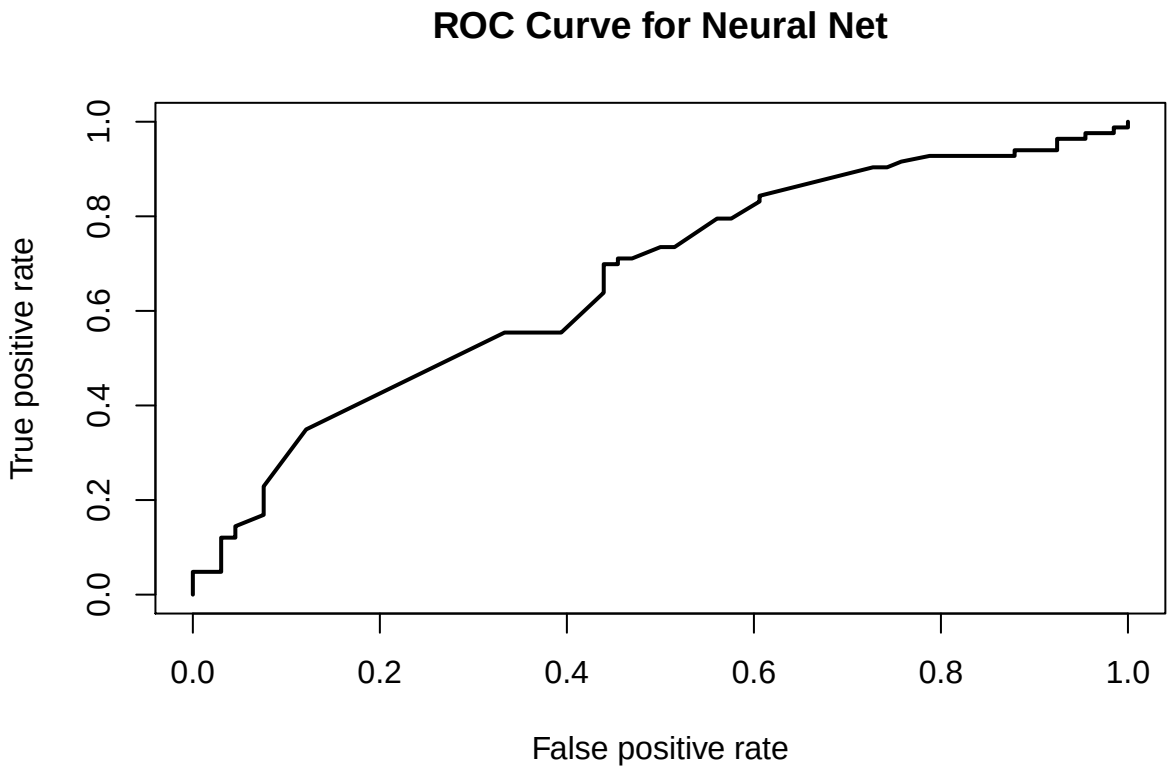



```
# Not a great looking ROC curve.
```

#Let's tune the NNet model!

```
# Make the tuning grid
trcontrol <- trainControl(method = "cv", number = 5, search = "grid",
    verboseIter = FALSE)
tunegrid = expand.grid(size = c(5:10), decay = seq(1e-04, 0.05,
    length.out = 20))
# Pass the tuning grid into the NN
nnet_tune <- train(Y_train2 ~ ., data = as.data.frame(cbind(X_train2,
    Y_train2)), method = "nnet", trControl = trcontrol, tuneGrid = tunegrid)
# Run the predictions with the tuned NN model
nnet.preds = predict(nnet_tune, newdata = X_test2, type = "prob")
nnet.pred = prediction(nnet.preds[, 2], Y_test2)
nnet.perf = performance(nnet.pred, "tpr", "fpr")
nnet.auc = performance(nnet.pred, "auc")
# Area under curve again
nnet.area = nnet.auc@y.values[[1]]
print(nnet.area)
# Small improvement at 0.6512 Build and print confusion
# matrix
Predicted = nnet.preds[, 2] >= 0.5
Actual = Y_test2
tabs.nnet <- table(Predicted, Actual)
dimnames(tabs.nnet)[[1]] <- c(levels(Y_train2)[1], levels(Y_train2)[2])
print(confusionMatrix(tabs.nnet, positive = "1"), mode = "everything")
```

```
# We gained an interesting trade off. The accuracy is now
# 0.6174, we're catching more true positives, but also
# missing a few true negatives. Overall, not as good as our
# tuned elastic net.  Actual Predicted 0 1 0 29 15 1 42 63

plot(nnet.perf, col = 1, lwd = 2, main = "ROC Curve for Neural Net")  #Curve still looks worse than our
```

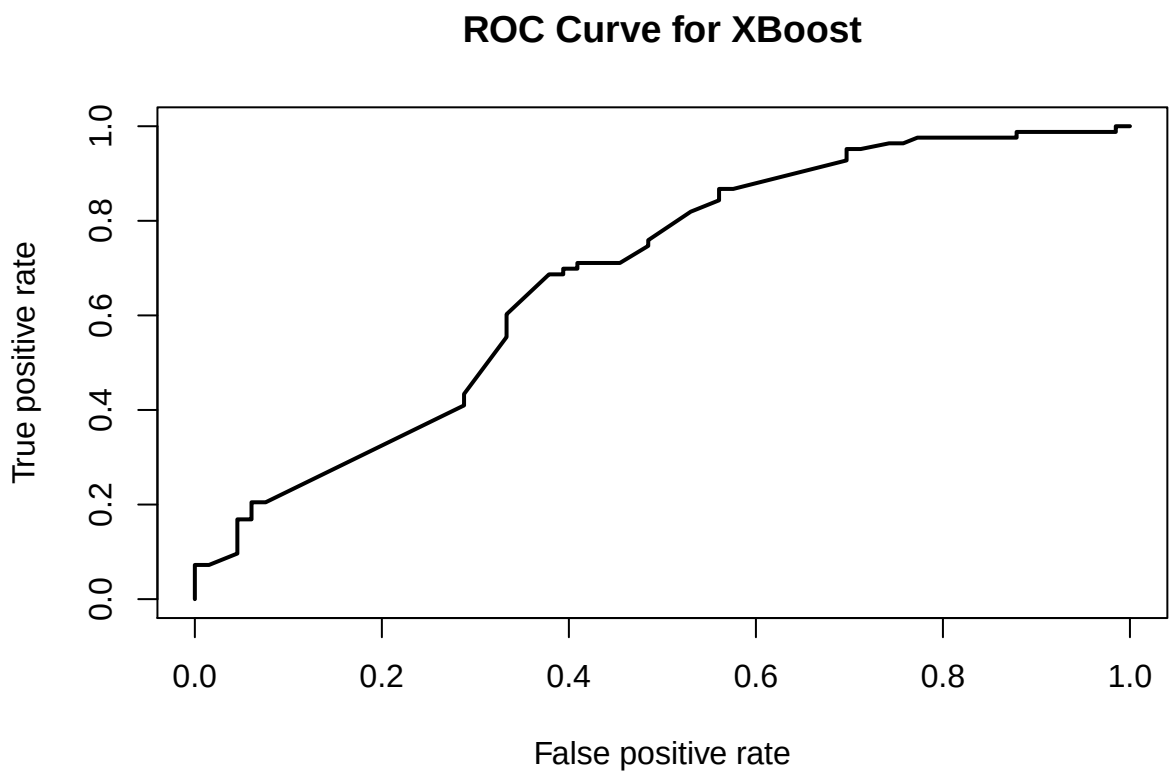


Let's try the Extreme Gradient Boosting (xgboost)

```
fitControl <- trainControl(method = "cv", number = 3, search = "grid",
    verboseIter = FALSE)

# Make the grid
XgbmGrid <- expand.grid(nrounds = c(100, 500), lambda = seq(0.05,
    200, length.out = 2), alpha = runif(3, 0, 1), eta = seq(1e-04,
    0.5, length.out = 3))
# Pass the tuning grid into xgboost model and the
# test/training data
Xgbc.fit <- train(as.factor(Y_train2) ~ ., data = cbind(Y_train2,
    X_train2), method = "xgbLinear", trControl = fitControl,
    tuneGrid = XgbmGrid, verbose = FALSE)
```

```
# Make the predictions with the xgbc model
xgbc.preds = predict(Xgbc.fit, newdata = X_test2, type = "prob")[,
    2]
```

```r
xgbc.pred = prediction(xgbc.preds, Y_test2)
xgbc.perf = performance(xgbc.pred, "tpr", "fpr")
xgbc.auc = performance(xgbc.pred, "auc")

# Get the area under curve again
xgbc.area = xgbc.auc@y.values[[1]]
print(xgbc.area)
# 0.668 Create and print the confusion matrix
Predicted = xgbc.preds >= 0.5
Actual = Y_test2
tabs.xgbc <- table(Predicted, Actual)
dimnames(tabs.xgbc)[[1]] <- c(levels(Y_train2)[1], levels(Y_train2)[2])

print(confusionMatrix(tabs.xgbc, positive = "1"), mode = "everything")
# Actual Predicted 0 1 0 30 14 1 41 64

# We gained another correctly classified company over the
# original winner that was the elasticnet logit, and we moved
# the company from a false positive, to a true negative.
# Overall we gained a minor boost in F1, which went from
# 0.6957 to 0.6995.

plot(xgbc.perf, col = 1, lwd = 2, main = "ROC Curve for XBoost")
```




```r
# Not a super strong ROC curve, probably room for improvement
# if we could get a better data set
```

#Conclusion #We attempted to fit various classification models to see if we can screen out leads that a VC firm should pass on instead of spending more time vetting manually using an analyst in due diligence. We encounted some limitations due to not having a lot of training data, but found that there's likely merit to the idea if we can get a more robust training and test set.