Optimizing Classification for Cost-Sensitive Decision Making: A Deep Learning Approach

DS 7333 Quantifying the World

David Shaw, Jeff Nguyen, David Julovich

April 15, 2021

1. Introduction

The data science team has been given an anonymized dataset encoding 50 features, with the express goal of classifying a target variable as accurately as possible. From the preliminary requirements gathering, the business team identified that false positives are far more costly than false negatives, by a factor of more than six. Features within the dataset were unimportant to the business team and that the transparency of the model was not important. Given these parameters, we provide the methods and analysis used to generate classification models where false positives and false negatives are minimized with the total dollar cost of misclassifications as low as possible. The business team loses $225 for every false positive signal and $35 for false negatives. This informs us that false classifications are costly, especially when we are overly optimistic about a positive class. Initially lighter weight models were explored, such as logistic regression and random forest classifiers. These are standard machine learning techniques, commonly used in such classification tasks. Due to the cost of misclassification and the default opaqueness of the dataset, there was little reason to not explore deep learning techniques to perform this classification. Given the sufficient data size and computing resources deep learning approach proved to provide enormous predictive accuracy of the target feature.

2. Methods

2.1. Data Preparation and Feature Creation

The dataset of interest had no meta data and contained 50 unlabeled features with 160,000 rows. While some inference could be deduced from the features within the dataset, such as day of week, month, country, and balance in dollars; 92% of features were numerical values where real-world significance could not be determined. Thus, creating a blinded dataset where domain knowledge could not be applied. Three approaches were taken to clean and prepare the data for modeling.

Approach 1 consisted of dropping all NAs and Nulls, categorical features where mutated into numeric levels, and strings features were converted to numeric values. Approach 1 produced a dense dataset that contained 50 features, 1 response variable and 158,392 rows. Approach 2 followed the same steps outlined in Approach 1 and then conducted column multiplication. This consisted of taking all values in one column and multiplying by the values of another column to create a new feature. This strategy was carried out across the data set using different combination of columns. Approach 2 produced a dataset with 1275 features, 1 response variable and 158,392 rows. The value of this approach is that the model may potentially capture a strong relationship between a pairwise feature and the target that is not present in either feature alone.

Approaches 1 and 2 reduced the number of records using case wise deletion by only 1% when removing NAs and Nulls. This percent is very small percentage of the data and will not cause a loss power. Approach 3 imputed all NAs and Nulls to the value 0. Categorical features were one hot encoded and produced 30 additional features in a sparse format. Approach 3 produced a mixed dataset that contained 80 features, 1 response variable and 160,000 rows.

Each approach used python's standard scalar and then train test splits were created with a 20% hold out. An additional 3 cross fold validation was used on Model 3, the best performing model.

The different EDA approaches had varying results when run through the 3 ANN models keeping all parameters constant. Approach 1 consistently produced less false positives and false negatives which equated to a saving 10s of thousands of dollars over Approaches 2 and 3. From our analysis we conclude that a dense data set without imputation or feature engineering was superior.

2.2. Modeling

2.2.1. Model Selection

From requirements gathering, we were told that the sole requirement is minimizing the company's lost value from misclassifications. Given this requirement, the size of the dataset, and the computational resources available, a neural network approach is the optimal solution. Neural networks and deep learning approaches have the potential to find patterns in the target set by fitting thousands upon thousands of layered equations in order to capture non-linearities in the mapping between the features and the target. Given enough training data, neural networks boast state of the art performance that can only be beat by extremely solid domain expertise and more targeted data collection, which is not a luxury available to our team.

The chief downsides of deep learning models are firstly that a lot of data is needed, and secondly that they are computationally expensive to train. The first concern is not a problem here, we have at least 150,000 data points with which to train our model. The second concern is also mitigated – our team has had two weeks to produce this analysis, and we are able to leverage the local resources we have without needing an additional cloud computation budget.

2.2.2. Model Architecture

We selected a deep neural network approach as our approach to this classification task – the next step is to define our deep learning architecture. We have to select an appropriate number of layers; each layer is defined by a number of neurons, with an activation function and a weight. The ultimate goal of any neural network is to pass the appropriate signal to the output layer; for a binary classification task like this, the output layer is typically a single neuron that outputs a value from 0 to 1 indicating the likelihood of the signal being positive.

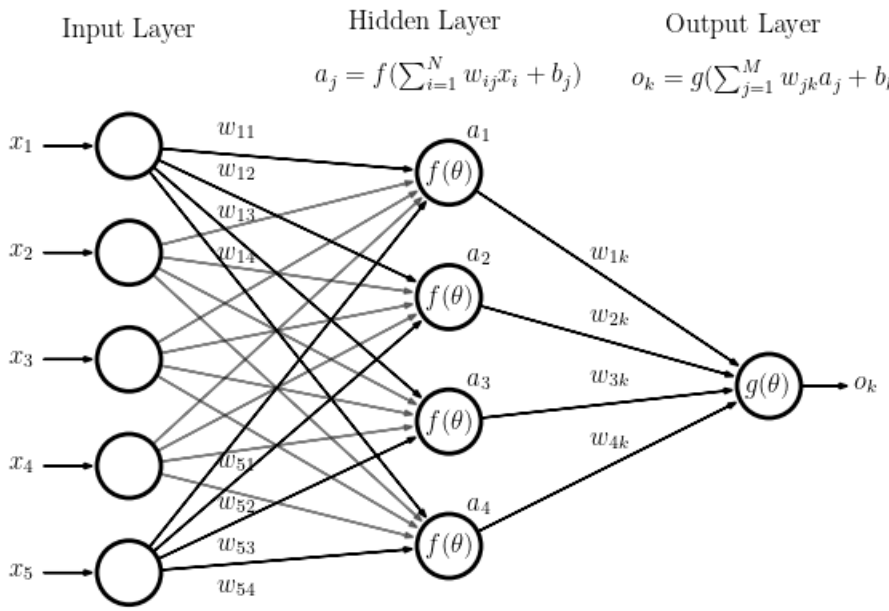

Fig 1. A basic neural network diagram [1]

For our model we utilized a 3-layer artificial neural network. Since there were 50 features a 3-layer architecture was selected since it gives a balance of performance and faster computation time. For all models, the initial layer contained 100 nodes, the second layer contained 55 nodes, and the last layer contained 1 node. The node selection for layers 1 and 2 were arbitrarily chosen and adhere to conventions for construction of ANNs. The last layer contained one node with a sigmoid function, this allows the layer to generate binary classifications from values passed in from previous layers.

A total of 4 models were constructed. Model 1 had three layers as described previously. Model 2 had the addition of 2 drop-out layers – one between each layer – that help improve the strength of strong signals and improve model performance. Each drop out layer was set to a value of 0.15. The third model was similar to the second model; however, the first dropout value was changed to 0.25 and the second dropout layer was changed to 0.15. Adjusting these values further improved model performance by weeding out weak signals present in the ANN. The fourth model was a 3-fold cross validation on the third model. This was preformed to see how well the third model would generalize to data – evidence of this property would be similar metrics between each fold.

Our models were optimized on precision and recall. These metrics were chosen as precision seeks to maximize true positives over all values predicted to be positive, high precision values in turn result in fewer false positives; recall seeks maximize true positives for all records that are positive, high recall values in turn result in fewer false positives. To find the optimal model three ANNs were constructed where parameters were manually adjusted. Once the best model was identified, a 3-fold cross validation was preformed to see how well the model generalizes. A model that performs well in the cross-validation is more likely to perform well on new data that the model has not been exposed to.

2.3. Model Evaluation

Importantly, unlike most classification problems, the difference between false positives and false negatives is very important to our problem. In most classification problems, the goal is to maximize metrics like the area under the receiver operating characteristic curve, which is an aggregate measure of how accurate the model is at predicting both positive and negative class instances. However, our true goal as stated is to minimize the dollar amount lost to misclassifications. Unlike most classification tasks, our cost function is asymmetrical - we would rather tolerate 6 false negatives than a single false positive. As such, we may need to adjust the classification threshold of the final model; for example, even if an output indicates that the class instance is 60% likely to be positive, the expected value of guessing negative is still higher than the expected value of guessing positive. We can tune this threshold by finding the value that minimizes our expected loss on the testing dataset, or the portion of data that was not used in training our model. This test dataset is supposed to represent a real case study of our model's performance on unseen data.

3. Results

Model	True Positive	False Positive ($225)	True Negative	False Negative ($35)	Precision	Recall	Monetary Loss
ANN Model 1	12080	581	18515	503	0.9541	0.9600	*$149,330*
ANN Model 2	12049	370	18726	534	0.9702	0.9576	*$101,940*
ANN Model 3	12041	240	18856	542	0.9805	0.9569	*$72,970*
ANN Model 3 CV	12116	253	18843	467	0.9795	0.9629	*$73,270*

Table 1: Initial Model Results at 0.5 cutoff

We can see as we progress through each iteration of ANN model, the precision improves, the recall is a little inconsistent but remains at a high value, and the monetary loss steadily decreases. This shows that the addition and tuning of the dropout layers markedly help improve the model. Model 3 was identified as the best model as it had the highest precision, despite the lower recall, this model had the lowest monetary loss at $72,970. The model was then subjected to a 3-fold cross validation where similar results were generated. This provides good evidence that Model 3 will be able to perform well on new data.

Model	True Positive	False Positive ($225)	True Negative	False Negative ($35)	Precision	Recall	Monetary Loss
ANN Model 3 CV	11911	150	18946	672	0.9876	0.9466	*$57,270*

Table 2: Model Results at 0.67 cutoff

We iteratively identified the ideal classification threshold by identifying each permutation of monetary loss for a given threshold from 0-100%. Using this method we identified that a classification threshold of 67% yielded the best results with a total monetary loss of $57,270. Table 2 illustrates the classification results as well as model metrics and the overall monetary loss.

4. Discussion

Without any data analysis or business intelligence, the company has a few options. It can either avoid costly false positives at all costs, and accept the $35 loss every time we have a positive class instance, or it can try and guess positives using a weighted coin flip. However, by incorporating some basic data science knowledge we can improve this baseline significantly. Assuming that our validation dataset is representative of real-world data, we have 12,061 positive class instances and 19,618 negative class instances, as illustrated in Table 2. Going by the first baseline approach, of always accepting negatives, the company stands to lose $422,135 – each of the 12,061 positive instances costs $35. On average, the company can expect to lose about $13.32 on each prediction, based on dividing this gross cost by the total number of instances in the validation set (31,679).

However, by leveraging data science and deep learning into business intelligence, we can reduce that gross cost to $57,270, or $1.81 per prediction. This is an average cost saving of 86%, which is tremendously valuable to the company given the frequency of predictions. If the company would like to start expanding its production of the product set and the number of predictions grows, then the cost savings will only increase with scale. Additionally, with more datapoints the power of our neural network model will only increase as well. Our conclusion is that employing a neural network to perform classification tasks in this space is fiscally prudent, and will pay enormous short and long-term dividends for the company.

5. References

1. https://www.astroml.org/book_figures/chapter9/fig_neural_network.html

6. Appendix

Fold Number	Loss	Precision	Recall
1	0.0881	0.9760	0.9607
2	0.0969	0.9732	0.9539
3	0.0914	0.9764	0.9578

Table 3: Model 3 CV Results

Table 3 describes the results for Model 3's cross validation results. This process allows us to see how well the model would generalize to new data. The consistent metrics between each fold suggest that this is the case and demonstrate strong precision and recall values.

R and Python Code

```
EDA in R
library(skimr)
library(dplyr)
getwd()
cs_14<-read.csv('C:/Users/daj0079/Desktop/smu2021/CS
14/final_project.csv')
skim(cs_14)
session-timeout-minutes=0
#droping all NAs only removes 160000-158511= 1489 rows
no_cs_14<-na.omit(cs_14)
skim(no_cs_14)
# set the target
target <- no_cs_14[c(51)]
# DROP THE Target from dataframe
no_cs_14<- no_cs_14[ -c(51) ]
skim(no_cs_14)
# Check data for column x24
unique(no_cs_14$x24)
# count the number of entries for each month
# "" has 30 rows not sure what do with these just yet
table(cs_14$x24)
# convert country name to "america"=1,"asia"=2,"euorpe"=3
no_cs_14$x24_country[no_cs_14$x24== "america"] <- 1
no_cs_14$x24_country[no_cs_14$x24== "asia"] <- 2
no_cs_14$x24_country[no_cs_14$x24== "euorpe"] <- 3
no_cs_14$x24_country[no_cs_14$x24== ""] <- 9999
no_cs_14$x24_country
sum(is.na(no_cs_14$x24_country))
table(cs_14$x24_country)
unique(no_cs_14$x24_country)
# DROP THE CATERGORICAL COL
no_cs_14<- no_cs_14[ -c(25) ]
# look at months
unique(no_cs_14$x29)
#NOTE THESE ARE MESSY
# "July"    "Aug"     "Jun"     "May"     "sept."   "Apr"     "Nov"
"Oct"     ""        "Mar"     "Feb"
# "Dev"     "January"
 # count the number of entries for each month
# "" has 30 rows not sure what do with these just yet
table(cs_14$x29)
#Set the entries with no month to the 13th month for now
# maybe able to determine actual month later
no_cs_14$x29_month[no_cs_14$x29== "January"] <- 1
no_cs_14$x29_month[no_cs_14$x29== "Feb"] <- 2
no_cs_14$x29_month[no_cs_14$x29== "Mar"] <- 3
no_cs_14$x29_month[no_cs_14$x29== "Apr"] <- 4
no_cs_14$x29_month[no_cs_14$x29== "May"] <- 5
no_cs_14$x29_month[no_cs_14$x29== "Jun"] <- 6
```

```
no_cs_14$x29_month[no_cs_14$x29== "July"] <- 7
no_cs_14$x29_month[no_cs_14$x29== "Aug"] <- 8
no_cs_14$x29_month[no_cs_14$x29== "sept."] <- 9
no_cs_14$x29_month[no_cs_14$x29== "Oct"] <- 10
no_cs_14$x29_month[no_cs_14$x29== "Nov"] <- 11
no_cs_14$x29_month[no_cs_14$x29== "Dev"] <- 12
no_cs_14$x29_month[no_cs_14$x29== ""] <- 9999
# confirm everything looks good for x29_month
no_cs_14$x29_month
sum(is.na(no_cs_14$x29_month))
unique(no_cs_14$x29_month)
# DROP THE CATERGORICAL COL
no_cs_14<- no_cs_14[ -c(29) ]
# days of week
#"tuesday"   "wednesday" "thurday"   "monday"    "friday"    ""
unique(cs_14$x30)
#"" has 30 rows not sure what do with these just yet
table(cs_14$x30)
#Set the entries with no DAY to the 6th day for now
# maybe able to determine actual day later
no_cs_14$x30_day[no_cs_14$x30== "monday"] <- 1
no_cs_14$x30_day[no_cs_14$x30== "tuesday"] <- 2
no_cs_14$x30_day[no_cs_14$x30== "wednesday"] <- 3
no_cs_14$x30_day[no_cs_14$x30== "thurday"] <- 4
no_cs_14$x30_day[no_cs_14$x30== "friday"] <- 5
no_cs_14$x30_day[no_cs_14$x30== ""] <- 9999
# confirm everything looks good for x29_month
no_cs_14$x30_day
sum(is.na(no_cs_14$x30_day))
unique(no_cs_14$x30_day)
# DROP THE CATERGORICAL COL
no_cs_14<- no_cs_14[ -c(29) ]
# float that need % and quote removed
# "0.00%"  "-0.02%" "-0.01%" "0.01%"  "-0.03%" "0.02%"  "-0.04%" ""
"0.03%"  "0.04%"  "-0.05%" "0.05%"
unique(cs_14$x32)
# count the number of entries for each month
# "" has 30 rows not sure what do with these just yet
table(cs_14$x32)
#Set the entries with no month to the 13th month for now
# maybe able to determine actual month later
no_cs_14$x32_percent[no_cs_14$x32== "0.00%"] <- 0
no_cs_14$x32_percent[no_cs_14$x32== "-0.02%"] <- -0.02
no_cs_14$x32_percent[no_cs_14$x32== "-0.01%"] <- -0.01
no_cs_14$x32_percent[no_cs_14$x32== "0.01%"] <- 0.01
no_cs_14$x32_percent[no_cs_14$x32== "-0.03%"] <- -0.03
no_cs_14$x32_percent[no_cs_14$x32== "0.02%"] <- 0.02
no_cs_14$x32_percent[no_cs_14$x32== "-0.04%"] <- -0.04
no_cs_14$x32_percent[no_cs_14$x32== "0.03%"] <- 0.03
no_cs_14$x32_percent[no_cs_14$x32== "0.04%"] <- 0.04
no_cs_14$x32_percent[no_cs_14$x32== "-0.05%"] <- -0.05
no_cs_14$x32_percent[no_cs_14$x32== "0.05%"] <- 0.05
```

```
no_cs_14$x32_percent[no_cs_14$x32== ""] <- 9999
# confirm everything looks good for x29_month
no_cs_14$x32_percent
sum(is.na(no_cs_14$x32_percent))
unique(no_cs_14$x32_percent)
# DROP THE CATERGORICAL COL
no_cs_14<- no_cs_14[ -c(30) ]
sum(is.na(target))
sum(is.na(no_cs_14))
skim(no_cs_14)
# Remove all rows that contained no data.
no_cs_14<- no_cs_14[!(no_cs_14$x24_country== 9999 |
no_cs_14$x29_month==9999| no_cs_14$x30_day == 9999|
no_cs_14$x32_percent ==9999),]
#save clean data set
write.csv(no_cs_14,"C:/Users/daj0079/Desktop/smu2021/QTW_Homework/CS_1
4/Full_data.csv")
library(GGally)
library(ggplot2)
no_cs_14$y <- as.factor(no_cs_14$y)
ggpairs(no_cs_14,columns = 1:50, ggplot2::aes(colour=y))
#Multiply columns to generate a data of 1276 columns
res<- cbind(no_cs_14^2, do.call(cbind, combn(colnames(no_cs_14), 2,
                       FUN = function(x)list(no_cs_14[x[1]] *
no_cs_14[x[2]]))))
colnames(res)[-(seq_len(ncol(no_cs_14)))]<-combn(colnames(no_cs_14),
2, FUN= paste, collapse= ":")
install.packages("ggcorrplot")
library(ggcorrplot)
ggcorrplot(corr, hc.order = TRUE, type = "lower",
           lab = TRUE)
ggcorrplot(corr)
total<- cbind(no_cs_14, target)
# column to keep reduced features x2, x37_dollars, x41, x38, x49, x27,
x12, x46, x24_country, x20,x23,x48, x32_percent,x28, x7, x40
myvars <- names(no_cs_14) %in% c("x2", "x37_dollars", "x41", "x38",
"x49",
                                 "x27", "x12", "x46", "x24_country",
"x20","x23","x48",
                                 "x32_percent","x28", "x7", "x40")
newdata <- no_cs_14[myvars]
corr<-cor(newdata)
ggcorrplot(corr, hc.order = TRUE, type = "lower",
           lab = TRUE)
##Two data set selected features and full dataset
totalnewdata<- cbind(newdata, target)
fulldata<-cbind(no_cs_14, target)
resdata<-cbind(res, target)
###GLM on only selected features from Coor plot
library(dplyr)
library(tidyverse)
library(caret)
```

```
#Split the data into training and test set
set.seed(123)
training.samples <- totalnewdata$y %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- totalnewdata[training.samples, ]
test.data <- totalnewdata[-training.samples, ]
# Build the model
mylogit <- glm(y ~ ., data = totalnewdata, family = "binomial")
summary(mylogit)
#full dataset glm
set.seed(123)
training.samples <- fulldata$y %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- fulldata[training.samples, ]
test.data <- fulldata[-training.samples, ]
# Build the model
mylogit <- glm(y ~ ., data = fulldata, family = "binomial")
summary(mylogit)
# added  features glm
set.seed(123)
training.samples <- resdata$y %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- resdata[training.samples, ]
test.data <- resdata[-training.samples, ]
# Build the model
mylogit <- glm(y ~ ., data = resdata, family = "binomial")
summary(mylogit)
# model accuracy
probabilities <- mylogit %>% predict(test.data, type = "response")
predicted.classes <- ifelse(probabilities > 0.5, "1", "0")
mean(predicted.classes==test.data$y)
pdata <- predict(mylogit, newdata = train.data, type = "response")
confusionMatrix(train.data , reference = newdata$y)
par(mfrow = c(2, 2))
plot(mylogit)
probabilities <- mylogit %>% predict(test.data, type = "response")
predicted.classes <- ifelse(probabilities > 0.5, "1", "0")
# Model accuracy
mean(predicted.classes==test.data$y)
#STEP WISE
library(MASS)
step.model <- mylogit %>% stepAIC(trace = FALSE)
coef(step.model)
probabilities <- step.model %>% predict(test.data, type = "response")
predicted.classes <- ifelse(probabilities > 0.5, "1", "0")
# Model accuracy
mean(predicted.classes==test.data$y)


# Logit and threshold exploration
library(data.table)
library(corrplot)
```

```
library(caret)

dt <- fread('~/Desktop/SMU/QTW/final_project.csv')


# x37 clearly a monetary column.
hist(as.numeric(substr(dt$x37, 2, 10000)))

# Appears normally distributed, centered around 0. Assign to dt
dt[, x37 := as.numeric(substr(x37, 2, 10000))]


# Missing values?
# create index
dt[, index := 1:nrow(dt)]
dt_no_nas = na.omit(dt)
na_dt = dt[setdiff(index, dt_no_nas$index)]

# Counts by column of nas
dt_no_nas[, lapply(.SD, function(x) sum(is.na(x)))]

# Not much pattern, similar distribution across columns.
# Every numeric column has at least 20 missing values - string columns
# and the target column do not have missing values.

# What did we learn from the imputation exercise? Let's just fill with
0's
for (j in seq_len(ncol(dt)))
set(dt,which(is.na(dt[[j]])),j,0)


# Normality tests
# Subsample since shapiro only handles <5000 records??
num_cols <- which(sapply(dt, is.numeric))
set.seed(12345)
subsample <- sample(dt$index, 5000)
dt[subsample, lapply(.SD, function(x) shapiro.test(x)$p.value),
.SDcols=num_cols]

# Deviation from normality in cols x7, x12, x20, x23, x28, x34, x40,
# x42, x46, and the target y
dt[, lapply(.(x7, x12, x20, x23, x28, x34, x40, x42, x46), hist)]

# Visually assessing - looks like normality

# Class balance?
dt[, .N, by='y']
# 95,803 negatives to 64,197 positives - 1.5 times more negatives

# First: establish a baseline.
# training split - 80 20 random split
set.seed(12345)
```

```
train_idxs = sample(dt$index, size=0.8 * nrow(dt))
test_idxs = setdiff(dt$index, train_idxs)
setkey(dt, 'index')

# What's our loss for no data science, only predicting negatives
observed = dt[test_idxs, y]
baseline_loss = sum(observed) * 35
# With no data science, we can just predict negative no matter since
it's safer.
# Doing so, we lose about $450,000 on our testing dataset.
# A good model will improve upon this metric


# Try a linear model? Lack of outliers and non-normality

# 225 for a false positive, 35 for a false negative
# ratio: need to be 6.43 times closer to positive to predict a
positive


model <- glm(y ~ ., family='binomial', data=dt[train_idxs, .SD,
.SDcols=!c('index')])
preds <- predict(model, dt[test_idxs, .SD, .SDcols=!c('index')])
# Transform to probability space
inv_logit <- function(x) exp(x) / (1+exp(x))
prob_preds = inv_logit(preds)

# What threshold minimizes loss?
# We shouldn't even bother looking under .6 probably
thresholds <- seq(.1, 1, by=.001)
losses <- sapply(thresholds, function(thresh) {
discrete_probs <- ceiling(prob_preds - thresh)
loss = sum(observed > discrete_probs) * 35 + sum(observed <
discrete_probs) * 225
})
plot(x=thresholds, y=losses, type = 'l')
best_loss <- min(losses)
improvement <- baseline_loss - best_loss
best_threshold <- thresholds[which.min(losses)]
# With a logistic regression, we can reduce the baseline cost by
$11,885
# Or by about 2.6%.
# overall accuracy at .5: also about 70%
accuracy <- function(preds, observed, threshold) {
sum(!xor(ceiling(preds - threshold), observed)) / length(observed)
}
plot(x=thresholds,
y=sapply(thresholds, accuracy, preds=prob_preds, observed=observed),
type='l')
```

```
# Python Modeling ###################################################
####################################################################
#%% Import Packages
import tensorflow as tf
import pandas as pd
import numpy as np
from tensorflow.keras import backend as K
from keras import Sequential
from tensorflow.keras import layers
from tensorflow.keras import initializers
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

#%% Read in data genrated by Dave's cleaning
df = pd.read_csv("C:/Users/Jeff/Desktop/Full_data.csv")

#%% Check df read in
df.head()

#%% Create train/test split

# Split
target = df['y']
df2 = df.drop(columns=['y'])
#scaler = StandardScaler()
#x_scaled = scaler.fit_transform(df2)

x_train, x_test,y_train, y_test = train_test_split(df2.values,
target.values, test_size=0.20, random_state=42)

####################################################################
####################################################################

#%% ANN 1st Model
model = tf.keras.Sequential()
model.add(layers.Dense(100,kernel_initializer=initializers.RandomNorma
l(stddev=0.05), activation = 'elu', input_shape = (50,)))
model.add(layers.Dense(55,kernel_initializer=initializers.RandomNormal
(stddev=0.01),activation = 'elu'))
model.add(layers.Dense(1,activation =
'sigmoid',kernel_initializer=initializers.RandomNormal(stddev=0.001)))

# Compile Model
opt = tf.keras.optimizers.Adam(learning_rate=0.001)

epochs = 500

model.compile(optimizer = opt,
              loss = 'binary_crossentropy',
              metrics = ['Precision','Recall']) #is there a better
optimizer or loss? tf.keras.metrics.AUC()
```

```
# Fit model
from tensorflow.keras.callbacks import EarlyStopping

es = EarlyStopping(monitor = 'val_loss', patience = 10) #set patience
higher if batch size is smaller

history = model.fit(x_train, y_train, epochs = epochs, validation_data
= (x_test,y_test),
                    batch_size = 100, callbacks = es)



#%% Predictions for 1st Model
preds = model.predict(x_test)
y_hat = (preds > 0.5)
y_hat = y_hat * 1

flatList = [ item for elem in y_hat for item in elem]
confusion_matrix(y_test, flatList)

######################################################################
######################################################################

#%% ANN 2nd Model


model2 = tf.keras.Sequential()
model2.add(layers.Dense(100,kernel_initializer=initializers.RandomNorm
al(stddev=0.05), activation = 'elu', input_shape = (50,)))
model2.add(layers.Dropout(0.15))
model2.add(layers.Dense(55,kernel_initializer=initializers.RandomNorma
l(stddev=0.01),activation = 'elu'))
model2.add(layers.Dropout(0.15))
model2.add(layers.Dense(1,activation =
'sigmoid',kernel_initializer=initializers.RandomNormal(stddev=0.001)))

# Compile Model
opt = tf.keras.optimizers.Adam(learning_rate=0.001)

epochs = 500

model2.compile(optimizer = opt,
               loss = 'binary_crossentropy',
               metrics = ['Precision','Recall']) #is there a better
optimizer or loss? tf.keras.metrics.AUC()


#%Fit model
from tensorflow.keras.callbacks import EarlyStopping
```

```
es = EarlyStopping(monitor = 'val_loss', patience = 10) #set patience
higher if batch size is smaller

history2 = model2.fit(x_train, y_train, epochs = epochs,
validation_data = (x_test,y_test),
                      batch_size = 100, callbacks = es)

#%% 2nd Model Predicitons
preds2 = model2.predict(x_test)
y_hat2 = (preds2 > 0.86)
y_hat2 = y_hat2 * 1

flatList2 = [ item for elem in y_hat2 for item in elem]

confusion_matrix(y_test, flatList2)
######################################################################
######################################################################

#%% ANN 3rd Model
model3a = tf.keras.Sequential()
model3a.add(layers.Dense(100,kernel_initializer=initializers.RandomNor
mal(stddev=0.05), activation = 'elu', input_shape = (50,)))
model3a.add(layers.Dropout(0.25))
model3a.add(layers.Dense(55,kernel_initializer=initializers.RandomNorm
al(stddev=0.01),activation = 'elu'))
model3a.add(layers.Dropout(0.15))
model3a.add(layers.Dense(1,activation =
'sigmoid',kernel_initializer=initializers.RandomNormal(stddev=0.001)))

# Compile Model

opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model3a.compile(optimizer = opt,
              loss = 'binary_crossentropy',
              metrics = ['Precision','Recall'])#is there a better
optimizer or loss? tf.keras.metrics.AUC()

# Fit model
from tensorflow.keras.callbacks import EarlyStopping

es = EarlyStopping(monitor = 'val_loss', patience = 50) #set patience
higher if batch size is smaller

history3 = model3a.fit(x_train, y_train, epochs = 500, validation_data
= (x_test,y_test),
                      batch_size = 100, callbacks = es)

#%% 3nd Model Predicitons

preds3a = model3a.predict(x_test)
y_hat3a = (preds3a > 0.5)
y_hat3a = y_hat3a * 1
```

```
flatList3a = [ item for elem in y_hat3a for item in elem]

confusion_matrix(y_test, flatList3a)


######################################################################
######################################################################

#%% Cross Validation

from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import KFold

kfold = KFold(n_splits = 3, shuffle=True)

# K-fold Cross Validation model evaluation
precision_per_fold = []
recall_per_fold = []
loss_per_fold = []
fold_no = 1
for train, test in kfold.split(df2, target):

    model3 = tf.keras.Sequential()

model3.add(layers.Dense(100,kernel_initializer=initializers.RandomNorm
al(stddev=0.05), activation = 'elu', input_shape = (50,)))
    model3.add(layers.Dropout(0.25))

model3.add(layers.Dense(55,kernel_initializer=initializers.RandomNorma
l(stddev=0.01),activation = 'elu'))
    model3.add(layers.Dropout(0.15))
    model3.add(layers.Dense(1,activation =
'sigmoid',kernel_initializer=initializers.RandomNormal(stddev=0.001)))

    #%% Compile Model
    opt = tf.keras.optimizers.Adam(learning_rate=0.001)

    model3.compile(optimizer = opt,
                   loss = 'binary_crossentropy',
                   metrics = ['Precision','Recall'])#is there a better
optimizer or loss? tf.keras.metrics.AUC()

    # Generate a print
    print('------------------------------------------------------------
-------------')
    print(f'Training for fold {fold_no} ...')

    # Fit data to model
```

```
    es = EarlyStopping(monitor = 'loss', patience = 50) #set patience
higher if batch size is smaller
    history3 = model3.fit(df2.iloc[train],
                      target[train],
                      epochs = 1000,
                      batch_size = 100,
                      callbacks = es)

    # Generate generalization metrics
    scores = model3.evaluate(df2.iloc[test], target[test], verbose=0)
    print(f'Score for fold {fold_no}: {model3.metrics_names[0]} of
{scores[0]}; {model3.metrics_names[1]} of {scores[1]*100}%')

    precision_per_fold.append(scores[1])
    recall_per_fold.append(scores[2])
    loss_per_fold.append(scores[0])

    # Increase fold number
    fold_no = fold_no + 1

######################################################################
######################################################################

#%% Performance Plots
import matplotlib.pyplot as plt

def performance_plots(model, epochs):
    f, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 4))
    t = f.suptitle('Neural Net Performance', fontsize=12)
    f.subplots_adjust(top=0.85, wspace=0.3)

    epoch_list = list(range(1,epochs+1))
    ax1.plot(epoch_list, model.history['precision'], label='Train
Precision')
    ax1.plot(epoch_list, model.history['val_precision'],
label='Validation Precision')
    ax1.set_xticks(np.arange(0, epochs+1, 100))
    #ax1.set_xticklabels(labels=np.arange(0, epochs+1,
25),rotation=30)
    ax1.set_ylabel('Precision Value')
    ax1.set_xlabel('Epoch')
    ax1.set_title('Precision')
    l1 = ax1.legend(loc="best")

    epoch_list3 = list(range(1,epochs+1))
    ax3.plot(epoch_list, model.history['recall'], label='Train
Recall')
    ax3.plot(epoch_list, model.history['val_recall'],
label='Validation Recall')
    ax3.set_xticks(np.arange(0, epochs+1, 100))
    #ax1.set_xticklabels(labels=np.arange(0, epochs+1,
25),rotation=30)
```

```
    ax3.set_ylabel('Recall Value')
    ax3.set_xlabel('Epoch')
    ax3.set_title('Recall')
    l3 = ax1.legend(loc="best")

    ax2.plot(epoch_list, model.history['loss'], label='Train Loss')
    ax2.plot(epoch_list, model.history['val_loss'], label='Validation
Loss')
    ax2.set_xticks(np.arange(0, epochs+1, 100))
    #ax2.set_xticklabels(labels=np.arange(0, epochs+1,
25),rotation=30)
    ax2.set_ylabel('Loss Value')
    ax2.set_xlabel('Epoch')
    ax2.set_title('Loss')
    l2 = ax2.legend(loc="best")


#%% 3nrd Model CV Predicitons
preds3 = model3.predict(x_test)


loss_list = []

for i in range(0,101):
    y_hat3 = (preds3 > i/100)
    y_hat3 = y_hat3 * 1

    flatList3 = [ item for elem in y_hat3 for item in elem]

    cm = confusion_matrix(y_test, flatList3)
    loss = cm[0][1] * 225 + cm[1][0] * 35
    loss_list.append(loss)

#%%
preds3 = model3.predict(x_test)
y_hat3 = (preds3 > 0.5)
y_hat3 = y_hat3a * 1

flatList3 = [ item for elem in y_hat3 for item in elem]

confusion_matrix(y_test, flatList3)
#%% Model 1
performance_plots(history, len(history.history['loss']))

#%% Model 2
performance_plots(history2, len(history2.history['loss']))

#%% Model 3
performance_plots(history3, len(history3.history['loss']))
```