Project 4

Code

Neural Networks

1 Processing the data

1.1 below we load the data

The data set we are using cleaned train dataset from project 1. We have 226434 observations and 35 variables in our df_train dataframe. The model we are using for finding the best performance is Neural Networks.

Hide

```
df_train <- readRDS("group5AA_Black-Boopathy_train_11_27.rds")
df_holdout<-readRDS("holdout_df.rds")
```

1.2 Releveling the data

We insisted the reference variable here is "Yes". For that, we use relevel function.To confirm our levels priority we use levels()function.

Hide

```
df_train$loan_default<-relevel(df_train$loan_default, ref= "Yes")
levels(df_train$loan_default)
```

```
## [1] "Yes" "No"
```

Hide

```
df_holdout$loan_default<-relevel(df_holdout$loan_default, ref="Yes")
levels(df_holdout$loan_default)
```

```
## [1] "Yes" "No"
```

1.3 pre processing recipe

Here, majority class is “No” with 182927 observations.For making a balanced samples we did Down sample majority to match minority and save the details in loan_prepped_train.us data frame and now my loan default in loan_prepped_train.us has balanced samples.

Hide

```
rec <- recipe(loan_default ~ ., data = df_train) %>%
  step_range(all_numeric_predictors(), min = 0, max = 1)
prep_loan <- prep(rec)

loan_prepped_train <- bake(prep_loan, new_data = df_train)
loan_prepped_holdout <- bake(prep_loan, new_data = df_holdout)

set.seed(123)
loan_prepped_train$loan_default <- as.factor(loan_prepped_train$loan_default)

loan_prepped_train.us=downSample(x=loan_prepped_train %>% dplyr::select(-loan_def
ault),
                      y=loan_prepped_train$loan_default,
                      yname="loan_default"
)

table(loan_prepped_train.us$loan_default)
```

```
##
##   Yes    No
## 43507 43507
```

Hide

```
table(loan_prepped_train$loan_default)
```

```
##
##    Yes     No
##  43507 182927
```

2 Basic nn Model

2.1 Run nn_model on the preprocessed Data

We Use the `nnet` method in `train()` to fit a single layer feed forward neural network using 10-fold cross validation. We Create a `tunegrid` to cross validate over the number of nodes in the hidden layer and the weight-decay. - We're telling R to consider hidden layer sizes of 1, 2, 3, 4, and 5 neurons.The parameter is L2 regularization parameter. Decay generates values from 0 to 0.01 in steps of 0.001(11 values). Now we have 55 rows in tune grid. We use parallel processing and caret model type (neural net from nnet package).we optimize on AUC(ROC).We suppress the output from training to printing using Trace=FALSE.

Hide

```
# tune_grid <- expand.grid(
#   size=c(1,2,3,4,5),
#   decay=seq(0,0.01,by=0.001)
# )
#
# ctrl <- trainControl(
#   method="cv",
#   number=10,
#   classProbs=TRUE,
#   summaryFunction=twoClassSummary
#
# )
#
# set.seed(123)
#
# # Set up Cluster
# cores=detectCores()
# cl=makeCluster(cores-1)
# registerDoParallel(cl)
#
# nn_model_1 <- train(
#   loan_default~.,
#   data=loan_prepped_train.us,
#   method="nnet",
#   trControl=ctrl,
#   tuneGrid=tune_grid,
#   metric="ROC",
#   maxit=500,
#   trace=FALSE
# )
#
# # Stop Cluster
# stopCluster(cl)
# registerDoSEQ()
#saveRDS(nn_model_1,"nn_us_model.rds")
nnet_model = readRDS("nn_us_model.rds")
```

We save the model and load it again to aviod rerunning multiple times.

2.2 Predict and Evaluate nn model

Now we are in the stage of prediction to evaluate the model performance with holdout data. For that, we use predict() function with nnet_model and loan_prepped_holdout to compute nnet_pred.prob and nnet_pred.class. We done our confusion matrix for analysing the tp,fn values will have resaonable predictions and checking the overall accuracy.

In continuation with this, we are creating the plot AUC metric. our AUC is 0.74 that's resonably better in catching tp against fp.

Hide

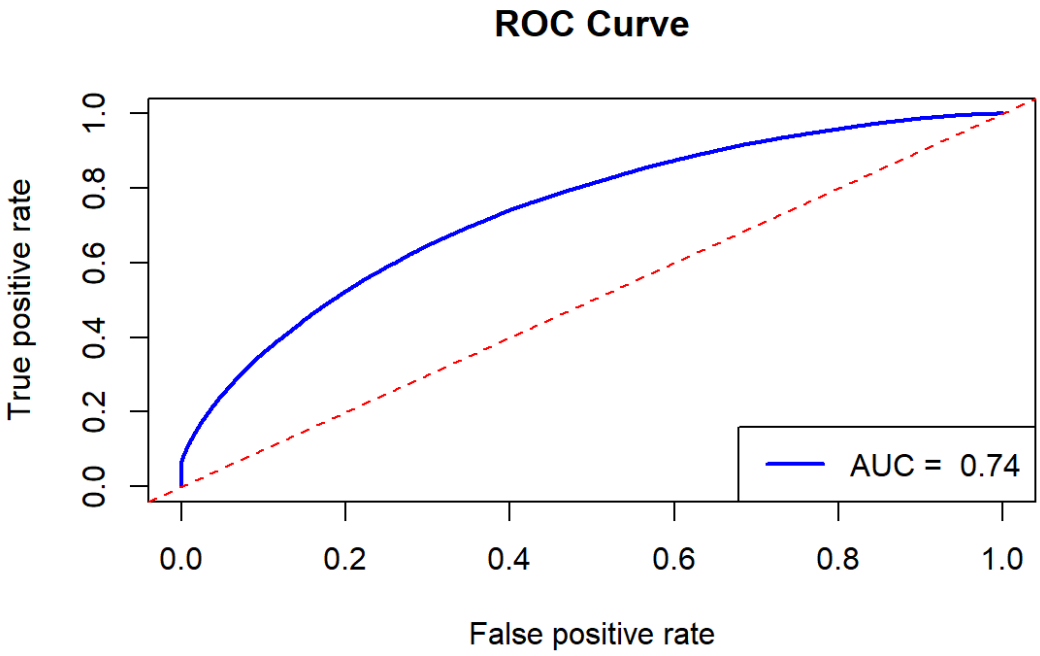
```
predmodels <- loan_prepped_holdout %>% dplyr::select(loan_default)
predmodels$nnet_pred.prob <- predict(nnet_model,newdata = loan_prepped_holdout,ty
pe="prob")[,"Yes"]
predmodels$nnet_pred.class <- predict(nnet_model, newdata = loan_prepped_holdout,
type = "raw")

confusionMatrix(predmodels$nnet_pred.class,predmodels$loan_default,positive="Ye
s")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    Yes     No
##        Yes  29009  58486
##        No   14498 124441
##
##                Accuracy : 0.6777
##                  95% CI : (0.6758, 0.6796)
##     No Information Rate : 0.8079
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2505
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6668
##             Specificity : 0.6803
##          Pos Pred Value : 0.3316
##          Neg Pred Value : 0.8957
##              Prevalence : 0.1921
##          Detection Rate : 0.1281
##    Detection Prevalence : 0.3864
##       Balanced Accuracy : 0.6735
##
##        'Positive' Class : Yes
##
```

Hide

```
probabilities=predict(nnet_model,newdata=loan_prepped_holdout,type="prob")
probabilities$actual=loan_prepped_holdout$loan_default
# Create Prediction Object
pred = prediction(probabilities[,"Yes"], loan_prepped_holdout$loan_default)
# Create Performance Object
perf = performance(pred, "tpr", "fpr")
auc = performance(pred,"auc")
auc_value=auc@y.values[[1]]
# Plot the ROC curve
plot(perf, col = "blue", lwd = 2, main = "ROC Curve")
abline(a=0, b=1, col="red", lty=2)
legend("bottomright",legend=paste("AUC = ",round(auc_value,3)),col="blue",lwd=2)
```



In this section, we use lasso model with logit fit. In project 3, we saved the logit function with the name “logit_model.rds”. we are going to utilize the function for finding the coefficients that are nonzero with boundaries -0.1 to 0.1 for strengthening the predictors effect and reducing the bias with low effect variables. Now the new distinction in this model is: we are using the selected predictors with target variables while splitting the train and test dataset and ensure those dataset has defined in selection. now our important variable has 14 in count.

3 NN Logit Model

3.1 Variable Selection from Logit Model

Hide

```
logit_fit=readRDS("logit_model.rds")

# Get the variable names
coefs_matrix=coef(logit_fit$finalModel,s=logit_fit$bestTune$lambda)
coefs=as.vector(coefs_matrix)
nonzero_idx=which(abs(coefs) > 0.1)
print(nonzero_idx)
```

```
##  [1]  1 12 16 17 18 19 21 22 23 24 25 26 28 29 31
```

Hide

```
logit_names=rownames(coefs_matrix)[nonzero_idx]
logit_names=logit_names[logit_names != "(Intercept)"]

options(scipen=999)
print(round(coef(logit_fit$finalModel,logit_fit$bestTune$lambda),4))
```

```
## 35 x 1 sparse Matrix of class "dgCMatrix"
##                                       s=0.001
## (Intercept)                          -2.9040
## loan_amnt                             0.0000
## int_rate                             -0.0223
## annual_inc                            0.0000
## dti                                  -0.0301
## open_acc                             -0.0176
## pub_rec                              -0.0820
## revol_bal                             0.0000
## revol_util                           -0.0007
## total_acc                             0.0111
## mort_acc                              0.0335
## pub_rec_bankruptcies                  0.1278
## fico_range_low                        0.0034
## fico_range_high                       0.0035
## inq_last_6mths                       -0.0810
## term__60_months                      -0.5345
## home_ownership_OWN                   -0.1552
## home_ownership_RENT                  -0.2554
## verification_status_Source_Verified -0.1080
## verification_status_Verified         -0.0024
## grade_B                              -0.2965
## grade_C                              -0.6329
## grade_D                              -0.8115
## grade_E                              -0.9365
## grade_F                              -1.0181
## grade_G                              -1.0509
## purpose_debt_consolidation           -0.0931
## purpose_home_improvement             -0.1919
## purpose_other                        -0.2169
## initial_list_status_w                 0.0273
## debt_settlement_flag_Y               -5.3471
## emp_length_4_6_years                  0.0395
## emp_length_7_9_years                  0.0223
## emp_length_10plus_years               0.0457
## interaction_dti_interest              0.0002
```

Hide

```
print(logit_names)
```

```
##  [1] "pub_rec_bankruptcies"                "term__60_months"
##  [3] "home_ownership_OWN"                  "home_ownership_RENT"
##  [5] "verification_status_Source_Verified" "grade_B"
##  [7] "grade_C"                             "grade_D"
##  [9] "grade_E"                             "grade_F"
## [11] "grade_G"                             "purpose_home_improvement"
## [13] "purpose_other"                       "debt_settlement_flag_Y"
```

Hide

```
log_train=loan_prepped_train.us[,c(logit_names,"loan_default")]

log_test=loan_prepped_holdout[,c(logit_names,"loan_default")]

table(log_train$loan_default)
```

```
##
##   Yes    No
## 43507 43507
```

3.2 Fit the nn_log Model

We Use the `nnet` method in `train()` to fit a single layer feed forward neural network using 10-fold cross validation. We Create a `tunegrid` to cross validate over the number of nodes in the hidden layer and the weight-decay. - We're telling R to consider hidden layer sizes of 1, 2, 3, 4, and 5 neurons.The parameter is L2 regularization parameter. Decay generates values from 0 to 0.01 in steps of 0.001(11 values). Now we have 55 rows in tune grid. We use parallel processing and caret model type (neural net from nnet package).we optimize on AUC(ROC) and maximum iterations has 500.We suppress the output from training to printing using Trace=FALSE.the dataset we are giving is log_train that is from previous codes.

After running the model, we saved as "nn_log_model.rds" for making the processing time efficiently.

Hide

```
# tune_grid <- expand.grid(
#   size=c(1,2,3,4,5),
#   decay=seq(0,0.01,by=0.001)
# )
#
# ctrl <- trainControl(
#   method="cv",
#   number=10,
#   classProbs=TRUE,
#   summaryFunction=twoClassSummary
#
# )
#
# set.seed(123)
#
# # Set up Cluster
# cores=detectCores()
# cl=makeCluster(cores-1)
# registerDoParallel(cl)
#
# nn_logmodel <- train(
#   loan_default~.,
#   data=log_train,
#   method="nnet",
#   trControl=ctrl,
#   tuneGrid=tune_grid,
#   metric="ROC",
#   maxit=500,
#   trace=FALSE
# )
#
# # Stop Cluster
# stopCluster(cl)
# registerDoSEQ()

#saveRDS(nn_logmodel,"nn_log_model.rds")
nn_logmodel=readRDS("nn_log_model.rds")
```

3.3 Predict and Evaluate nn_log model

Make a data frame that contains the target variable from the holdout sample as well as the probability predictions from your best neural network model and confusion matrix and AUC.

Hide

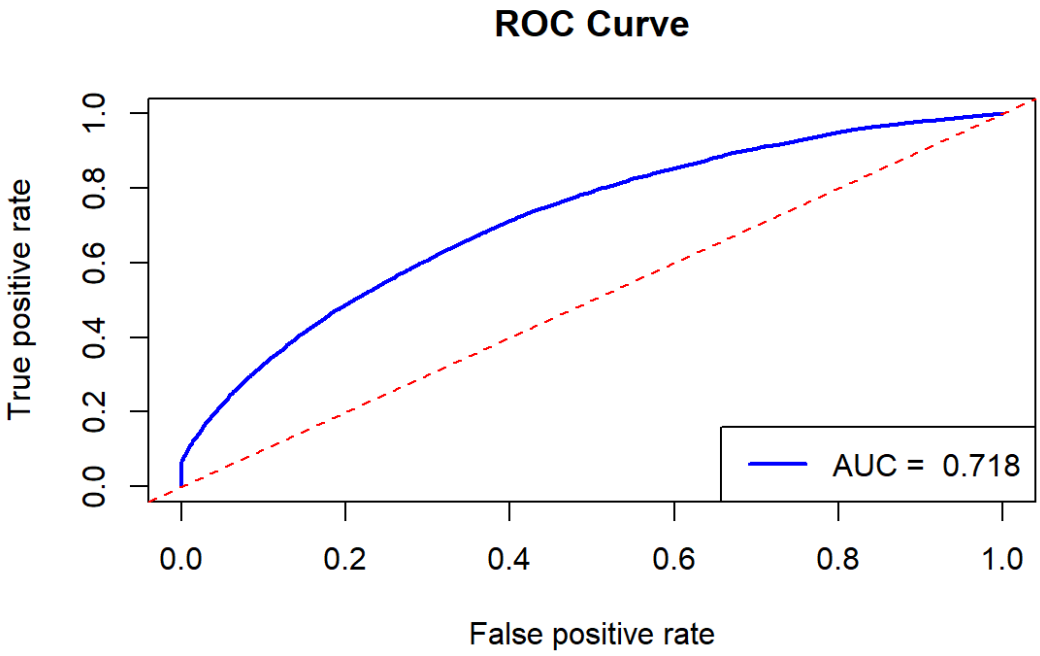
```
predmodels$nnet_log_pred.prob <- predict(nn_logmodel,newdata = log_test,type="pro
b")[,"Yes"]
predmodels$nnet_log_pred.class <- predict(nn_logmodel, newdata = log_test, type =
"raw")

confusionMatrix(predmodels$nnet_log_pred.class,predmodels$loan_default,positive
="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    Yes     No
##        Yes  29790  67979
##        No   13717 114948
##
##                Accuracy : 0.6392
##                  95% CI : (0.6372, 0.6412)
##     No Information Rate : 0.8079
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2122
##
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.6847
##             Specificity : 0.6284
##          Pos Pred Value : 0.3047
##          Neg Pred Value : 0.8934
##              Prevalence : 0.1921
##          Detection Rate : 0.1316
##    Detection Prevalence : 0.4318
##       Balanced Accuracy : 0.6565
##
##        'Positive' Class : Yes
##
```

Hide

```
probabilities=predict(nn_logmodel,newdata=log_test,type="prob")
probabilities$actual=log_test$loan_default
# Create Prediction Object
pred = prediction(probabilities[,"Yes"], log_test$loan_default)
# Create Performance Object
perf = performance(pred, "tpr", "fpr")
auc = performance(pred,"auc")
auc_value=auc@y.values[[1]]
# Plot the ROC curve
plot(perf, col = "blue", lwd = 2, main = "ROC Curve")
abline(a=0, b=1, col="red", lty=2)
legend("bottomright",legend=paste("AUC = ",round(auc_value,3)),col="blue",lwd=2)
```



4 NN Model with Keras

4.1 Fitting the NN Model with Keras

We are now introducing the new package in nnmodel as Keras3 for using two hidden layers and checking the performance(AUC). For this we installed python environment in R also installed tensorflow and pROC.Activation Function Introduces non-linearity. We use ReLU method. Common Loss Function for Classification - Cross-Entropy (for binary classification). keras_model_sequential() is the constructor that creates a new sequential model object in keras. We need an nn_model object to attach layers in it.Input layer has number of features in x_train and 32, 64 neurons in the layer and output function has 1 neuron with activation function has sigmoid.

Now, the compilation part has weight updates Adam in training and crossentrophy for binary classification. final part is fit in the steps of - trains the model on training data. epochs = 20 → the model sees the entire training set 20 times. batch_size = 32 → updates weights after every 32 samples. validation_split = 0.2 → reserves 20% of training data for validation during training. history stores the training log (loss, accuracy, AUC per epoch). evaluate() runs the trained model on test set (data not seen during training). It returns the loss and all metrics. we make a data frame that contains the target variable from the holdout sample as well as the probability predictions from my best neural network model and confusion matrix.

Hide

```
# Load libraries
library(keras3)
library(tensorflow)
library(nnet)
library(pROC)

x_train <- data.matrix(loan_prepped_train.us[, setdiff(names(loan_prepped_train.u
s), "loan_default")])
x_test  <- data.matrix(loan_prepped_holdout[, setdiff(names(loan_prepped_holdou
t), "loan_default")])

y_train <- ifelse(loan_prepped_train.us$loan_default == "Yes", 1, 0)
y_test  <- ifelse(loan_prepped_holdout$loan_default == "Yes", 1, 0)

nn_model <- keras_model_sequential() |>
  layer_dense(units = 64, activation = "relu", input_shape = ncol(x_train)) |>
  layer_dropout(rate = 0.3) |>
  layer_dense(units = 32, activation = "relu") |>
  layer_dropout(rate = 0.2) |>
  layer_dense(units = 1, activation = "sigmoid")

nn_model |> compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = list(metric_auc(), "accuracy")
)

history <- nn_model |> fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 32,
  validation_split = 0.2
)
```

```
## Epoch 1/20
## 2176/2176 - 3s - 1ms/step - accuracy: 0.6752 - auc: 0.7041 - loss: 0.5961 - va
l_accuracy: 0.4587 - val_auc: 0.0000e+00 - val_loss: 0.7655
## Epoch 2/20
## 2176/2176 - 2s - 934us/step - accuracy: 0.6873 - auc: 0.7207 - loss: 0.5831 -
val_accuracy: 0.3725 - val_auc: 0.0000e+00 - val_loss: 0.8912
## Epoch 3/20
## 2176/2176 - 3s - 1ms/step - accuracy: 0.6900 - auc: 0.7228 - loss: 0.5813 - va
l_accuracy: 0.4063 - val_auc: 0.0000e+00 - val_loss: 0.8552
## Epoch 4/20
## 2176/2176 - 2s - 940us/step - accuracy: 0.6897 - auc: 0.7245 - loss: 0.5801 -
val_accuracy: 0.4320 - val_auc: 0.0000e+00 - val_loss: 0.8650
## Epoch 5/20
## 2176/2176 - 2s - 933us/step - accuracy: 0.6893 - auc: 0.7253 - loss: 0.5794 -
val_accuracy: 0.4801 - val_auc: 0.0000e+00 - val_loss: 0.7512
## Epoch 6/20
## 2176/2176 - 2s - 938us/step - accuracy: 0.6907 - auc: 0.7263 - loss: 0.5789 -
val_accuracy: 0.4016 - val_auc: 0.0000e+00 - val_loss: 0.8518
## Epoch 7/20
## 2176/2176 - 2s - 930us/step - accuracy: 0.6913 - auc: 0.7277 - loss: 0.5777 -
val_accuracy: 0.4100 - val_auc: 0.0000e+00 - val_loss: 0.8562
## Epoch 8/20
## 2176/2176 - 2s - 919us/step - accuracy: 0.6920 - auc: 0.7279 - loss: 0.5775 -
val_accuracy: 0.4400 - val_auc: 0.0000e+00 - val_loss: 0.8398
## Epoch 9/20
## 2176/2176 - 2s - 979us/step - accuracy: 0.6923 - auc: 0.7287 - loss: 0.5767 -
val_accuracy: 0.4761 - val_auc: 0.0000e+00 - val_loss: 0.7935
## Epoch 10/20
## 2176/2176 - 2s - 929us/step - accuracy: 0.6898 - auc: 0.7291 - loss: 0.5765 -
val_accuracy: 0.4457 - val_auc: 0.0000e+00 - val_loss: 0.8157
## Epoch 11/20
## 2176/2176 - 2s - 1ms/step - accuracy: 0.6912 - auc: 0.7288 - loss: 0.5765 - va
l_accuracy: 0.4096 - val_auc: 0.0000e+00 - val_loss: 0.8531
## Epoch 12/20
## 2176/2176 - 2s - 955us/step - accuracy: 0.6919 - auc: 0.7298 - loss: 0.5762 -
val_accuracy: 0.4298 - val_auc: 0.0000e+00 - val_loss: 0.8125
## Epoch 13/20
## 2176/2176 - 2s - 933us/step - accuracy: 0.6936 - auc: 0.7303 - loss: 0.5759 -
val_accuracy: 0.4540 - val_auc: 0.0000e+00 - val_loss: 0.7881
## Epoch 14/20
## 2176/2176 - 2s - 938us/step - accuracy: 0.6933 - auc: 0.7308 - loss: 0.5754 -
val_accuracy: 0.4237 - val_auc: 0.0000e+00 - val_loss: 0.8276
## Epoch 15/20
## 2176/2176 - 2s - 927us/step - accuracy: 0.6929 - auc: 0.7307 - loss: 0.5753 -
val_accuracy: 0.4080 - val_auc: 0.0000e+00 - val_loss: 0.8257
## Epoch 16/20
```

```
## 2176/2176 - 2s - 925us/step - accuracy: 0.6944 - auc: 0.7319 - loss: 0.5744 -
val_accuracy: 0.4042 - val_auc: 0.0000e+00 - val_loss: 0.8350
## Epoch 17/20
## 2176/2176 - 3s - 1ms/step - accuracy: 0.6929 - auc: 0.7312 - loss: 0.5750 - va
l_accuracy: 0.4056 - val_auc: 0.0000e+00 - val_loss: 0.8207
## Epoch 18/20
## 2176/2176 - 2s - 948us/step - accuracy: 0.6941 - auc: 0.7320 - loss: 0.5742 -
val_accuracy: 0.4165 - val_auc: 0.0000e+00 - val_loss: 0.8320
## Epoch 19/20
## 2176/2176 - 2s - 938us/step - accuracy: 0.6941 - auc: 0.7324 - loss: 0.5737 -
val_accuracy: 0.3864 - val_auc: 0.0000e+00 - val_loss: 0.8289
## Epoch 20/20
## 2176/2176 - 2s - 931us/step - accuracy: 0.6941 - auc: 0.7327 - loss: 0.5737 -
val_accuracy: 0.4336 - val_auc: 0.0000e+00 - val_loss: 0.8280
```

Hide

```
keras_results <- nn_model |> evaluate(x_test, y_test)
```

```
## 7077/7077 - 4s - 597us/step - accuracy: 0.5146 - auc: 0.7377 - loss: 0.7491
```

Hide

```
#print(keras_results)
```

4.2 Model Prediction and Evaluation

Hide

```
# Keras ROC
nn_pred <- nn_model |> predict(x_test)
```

```
## 7077/7077 - 3s - 358us/step
```

Hide

```
roc_nn <- roc(y_test, nn_pred)
auc_nn <- auc(roc_nn)

print(paste("Keras AUC:", round(auc_nn, 3)))
```

```
## [1] "Keras AUC: 0.738"
```

Hide

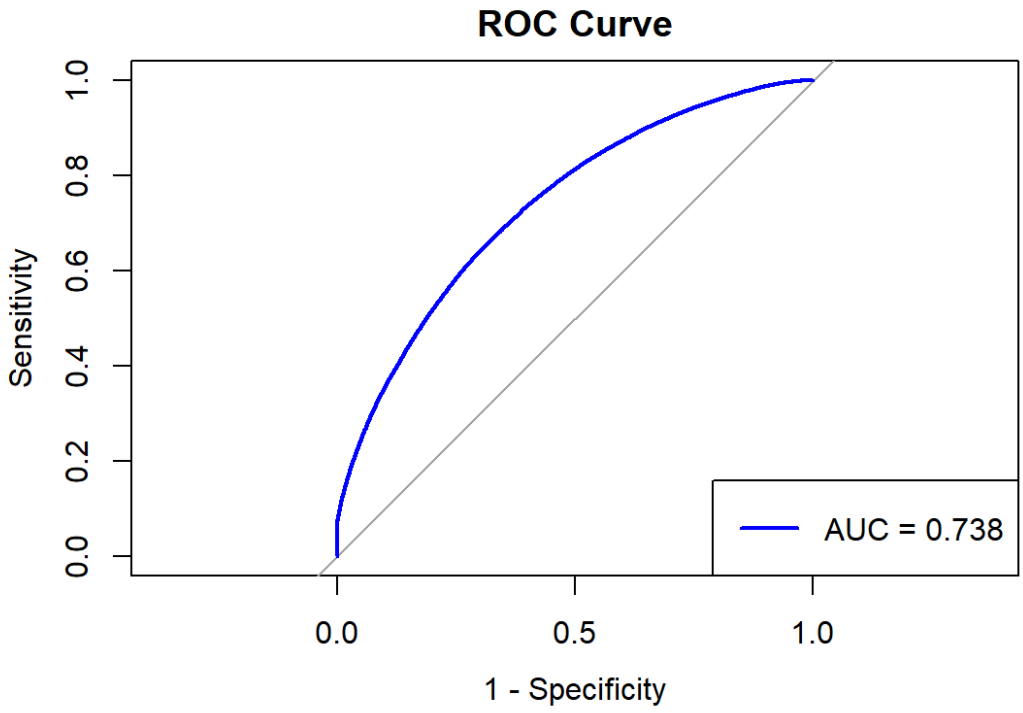
```
# Predictions
predmodels$nnet_keras.prob <- nn_pred
predmodels$nnet_keras.class <- factor(ifelse(predmodels$nnet_keras.prob > 0.5,
                                       "Yes","No"),
                                levels=c("Yes","No"))

confusionMatrix(predmodels$nnet_keras.class,predmodels$loan_default,positive="Ye
s")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    Yes     No
##        Yes  37191 103589
##        No    6316  79338
##
##                Accuracy : 0.5146
##                  95% CI : (0.5126, 0.5167)
##     No Information Rate : 0.8079
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1558
##
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.8548
##             Specificity : 0.4337
##          Pos Pred Value : 0.2642
##          Neg Pred Value : 0.9263
##              Prevalence : 0.1921
##          Detection Rate : 0.1642
##    Detection Prevalence : 0.6217
##       Balanced Accuracy : 0.6443
##
##        'Positive' Class : Yes
##
```

Hide

```
# Plot ROC curves
plot(roc_nn, col = "blue", lwd = 2, main = "ROC Curve", legacy.axes = TRUE)
legend("bottomright", legend = paste("AUC =", round(auc_nn, 3)), col = "blue", lw
d = 2)
```



5 AUC Metric Comparision for the Models

This code evaluates and compares the performance of three models (a neural network, a logistic regression, and a keras neural network) using ROC curves, AUC values and plot all the models performanace for choosing the better one and analysing the “tpr” against “fpr” .By seeing the AUC plot of all the three models the nn model has reaching the highest AUC of 0.74(74% of capturing “tpr” against “fpr”).

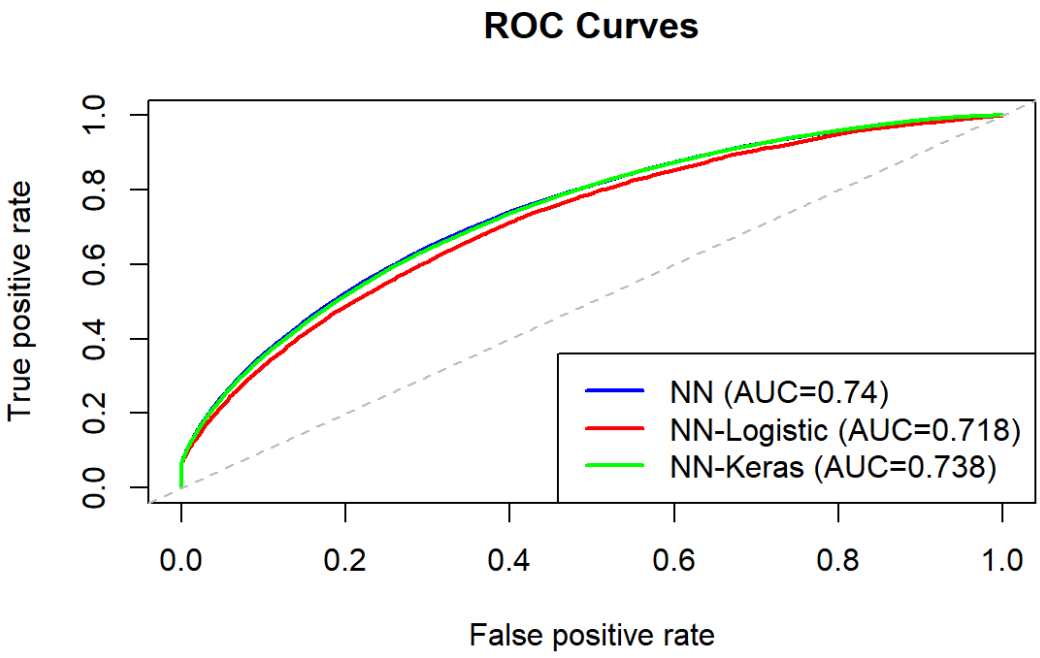
Hide

```
#Create prediction object
pred.nn=prediction(predmodels$nnet_pred.prob, predmodels$loan_default)
pred.nn_logistic=prediction(predmodels$nnet_log_pred.prob, predmodels$loan_defaul
t)
pred.nn_keras=prediction(predmodels$nnet_keras.prob, y_test)

# Create performance object
perf.nn=performance(pred.nn, "tpr", "fpr")
perf.nn_logistic=performance(pred.nn_logistic, "tpr", "fpr")
perf.nn_keras=performance(pred.nn_keras, "tpr", "fpr")

# Create AUC
auc.nn=performance(pred.nn, "auc")
auc.nn_logistic=performance(pred.nn_logistic, "auc")
auc.nn_keras=performance(pred.nn_keras, "auc")

plot(perf.nn, col="blue", lwd=2, main="ROC Curves")
plot(perf.nn_logistic, col="red", lwd=2, add=TRUE)
plot(perf.nn_keras, col="green", lwd=2, add=TRUE)
abline(a=0, b=1, col="gray", lty=2)
legend("bottomright",
       legend = c(
         paste0("NN (AUC=", round(auc.nn@y.values[[1]], 3), ")"),
         paste0("NN-Logistic (AUC=", round(auc.nn_logistic@y.values[[1]], 3),
")"),
         paste0("NN-Keras (AUC=", round(auc.nn_keras@y.values[[1]], 3), ")")
       ),
       col = c("blue", "red", "green"),
       lwd = 2)
```



6 Tuning the Predictions for threshold Using Youden's J statistic

This code identifies the optimal classification threshold for a neural network model using Youden's J statistic and then evaluates the model's performance with a confusion matrix:

Hide

```
# Calculate Youden's J
fpr=perf.nn@x.values[[1]]
tpr=perf.nn@y.values[[1]]
cutoffs=perf.nn@alpha.values[[1]]
j=tpr-fpr
# Find cutoff
best_j_idx=which.max(j)
best_j=j[best_j_idx]
best_thresh=cutoffs[best_j_idx]
# Print the best threshold and F1 score
cat("Best Threshold:", round(best_thresh, 3))
```

```
## Best Threshold: 0.511
```

Hide

```
# Use best threshold to classify
predmodels$nn_final = ifelse(predmodels$nnet_pred.prob > best_thresh, "Yes", "N
o") %>%
  factor(levels = c("Yes", "No"))

# Confusion matrix
confusionMatrix(predmodels$nn_final, predmodels$loan_default,positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    Yes     No
##        Yes  28242  55116
##        No   15265 127811
##
##                Accuracy : 0.6892
##                  95% CI : (0.6873, 0.6911)
##     No Information Rate : 0.8079
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2578
##
##  Mcnemar's Test P-Value : <0.0000000000000002
##
##             Sensitivity : 0.6491
##             Specificity : 0.6987
##          Pos Pred Value : 0.3388
##          Neg Pred Value : 0.8933
##              Prevalence : 0.1921
##          Detection Rate : 0.1247
##    Detection Prevalence : 0.3681
##       Balanced Accuracy : 0.6739
##
##        'Positive' Class : Yes
##
```

7 Final Analysis

By comparing all the three models, basic nn_model and nnmodel with keras have AUC 74%. But while considering the context of loan default we have chosen the model based on confusion matrix and accuracy value. The reason for using youdens j statistic is, if we see our confusion matrix of nn_model my tp has 29009 and fn has 14498 and 67.77% accuracy. In the loan default scenario, minimizing fn and increasing tp is the target. so if i am using the youden's J statistic my confusion

matrix reveals the best posssible values in finding fn and tp are 15265 and 28242 with 68.92% accuracy. so for achieving the better results i choose this probabilty would be the best in performance.

Using the youden's J in basic nn_network model my AUC has 74%.

So we would suggest considering basic nn_model for achieving the goal in loan_default.