R Lab: Evaluating Classification Models

In this lab, we evaluate the trained dense feed-forward neural network classification model from the previous lab using a Receiver Operating Characteristic (ROC) curve, area under the ROC curve (AUC ROC), and a calibration curve.

The deliverable for this lab is a Word or PDF file containing responses to the exercises at the end of the lab. The deliverable should be submitted through Canvas.

Data Pre-Processing and Neural Network Model Training

We first run the code from the previous labs to pre-process the raw data and train the dense feed-forward neural network model. Note, we use a dense feed-forward neural network model with two hidden layers (50 units in the first hidden layer and 25 in the second) and only train the model for 40 epochs, since the accuracy and validation loss appeared to get worse after approximately 40 epochs.

```
library(dplyr)
library(caret)
library(reticulate)
library(tensorflow)
library(keras)
library(MESS)

data <- read.csv("lab_5_data.csv")
training_ind <- createDataPartition(data$lodgepole_pine,
                                    p = 0.75,
                                    list = FALSE,
                                    times = 1)
training_set <- data[training_ind, ]
test_set <- data[-training_ind, ]

top_20_soil_types <- training_set %>%
  group_by(soil_type) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  select(soil_type) %>%
  top_n(20)

training_set$soil_type <- ifelse(training_set$soil_type %in% top_20_soil_types$soil_type,
                                 training_set$soil_type,
                                 "other")

training_set$wilderness_area <- factor(training_set$wilderness_area)
training_set$soil_type <- factor(training_set$soil_type)

onehot_encoder <- dummyVars(~ wilderness_area + soil_type,
                            training_set[, c("wilderness_area", "soil_type")],
                            levelsOnly = TRUE,
                            fullRank = TRUE)
```

```
onehot_enc_training <- predict(onehot_encoder,
                               training_set[, c("wilderness_area", "soil_type")])

training_set <- cbind(training_set, onehot_enc_training)

test_set$soil_type <- ifelse(test_set$soil_type %in% top_20_soil_types$soil_type,
                             test_set$soil_type,
                             "other")

test_set$wilderness_area <- factor(test_set$wilderness_area)
test_set$soil_type <- factor(test_set$soil_type)

onehot_enc_test <- predict(onehot_encoder, test_set[, c("wilderness_area", "soil_type")])
test_set <- cbind(test_set, onehot_enc_test)

test_set[, -c(11:13)] <- scale(test_set[, -c(11:13)],
                         center = apply(training_set[, -c(11:13)], 2, mean),
                         scale = apply(training_set[, -c(11:13)], 2, sd))
training_set[, -c(11:13)] <- scale(training_set[, -c(11:13)])

training_features <- array(data = unlist(training_set[, -c(11:13)]),
                  dim = c(nrow(training_set), 33))
training_labels <- array(data = unlist(training_set[, 13]),
                  dim = c(nrow(training_set)))

test_features <- array(data = unlist(test_set[, -c(11:13)]),
                  dim = c(nrow(test_set), 33))
test_labels <- array(data = unlist(test_set[, 13]),
                  dim = c(nrow(test_set)))

use_virtualenv("my_tf_workspace")

model <- keras_model_sequential(list(
  layer_dense(units = 50, activation = "relu"),
  layer_dense(units = 25, activation = "relu"),
  layer_dense(units = 1, activation = "sigmoid")
))

compile(model,
        optimizer = "rmsprop",
        loss = "binary_crossentropy",
        metrics = "accuracy")

history <- fit(model, training_features, training_labels,
    epochs = 40, batch_size = 512, validation_split = 0.33)
```

Evaluating Classification Models

Using the trained neural network model, we can predict the class membership for a new observation in the (held-out) test set. We use the 'predict' function to perform this prediction. The 'predict' function returns the probability of positive class membership, which is all we need for the evaluation metrics discussed here. For convenience, we will store these predictions in a new column of the test set, called 'p_prob', where the 'p'

stands for the 'positive' class.

```
predictions <- predict(model, test_features)
test_set$p_prob <- predictions[, 1]
head(predictions, 10)
```

```
##             [,1]
##  [1,] 0.8343194
##  [2,] 0.9204276
##  [3,] 0.7349368
##  [4,] 0.6033846
##  [5,] 0.9036730
##  [6,] 0.4538890
##  [7,] 0.4951064
##  [8,] 0.5464307
##  [9,] 0.1763184
## [10,] 0.4844534
```

ROC Curves

We can convert these probabilities to predictions using a threshold t: if 'p_prob' $>= t$, then predict class 1. Our problem is binary, so a natural threshold to use is 0.5, which corresponds to predicting the class with the largest probability.

```
over_threshold <- test_set[test_set$p_prob >= 0.5, ]
```

The dataframe 'over_threshold' holds the observations in the test set that have a 'p_prob' value over 0.5. Thus, if we were to use the natural threshold value of 0.5, the observations in 'over_threshold' would be predicted to belong to class 1. Let's calculate the false positive rate (FPR) and true positive rate (TPR) for these predictions. It is important to remember that class 1 is the positive class and class 0 is the negative class.

The FPR is the percentage of incorrect predictions we make for observations that belong to the negative class. Thus, we divide the number of observations in 'over_threshold' that belong to class 0 (since we are incorrectly predicting these to belong to class 1) by the total number of observations in the test set that belong to class 0.

```
fpr <- sum(over_threshold$lodgepole_pine==0)/sum(test_set$lodgepole_pine==0)
fpr
```

```
## [1] 0.3181395
```

The TPR is the percentage of correct predictions we make for observations that belong to the positive class. Thus, we divide the number of observations in 'over_threshold' that belong to class 1 (since we are correctly predicting these to belong to class 1) by the total number of observations in the test set that belong to class 1.

```
tpr <- sum(over_threshold$lodgepole_pine==1)/sum(test_set$lodgepole_pine==1)
tpr
```

```
## [1] 0.8197464
```

Thus, we can obtain a TPR of 0.8197464 at the cost of a FPR of 0.3181395. However, what if we vary this threshold for converting probabilities to predictions, instead of simply using 0.5? Using a threshold of 0.75 yields:

```
over_threshold <- test_set[test_set$p_prob >= 0.75, ]

fpr <- sum(over_threshold$lodgepole_pine==0)/sum(test_set$lodgepole_pine==0)
fpr
```

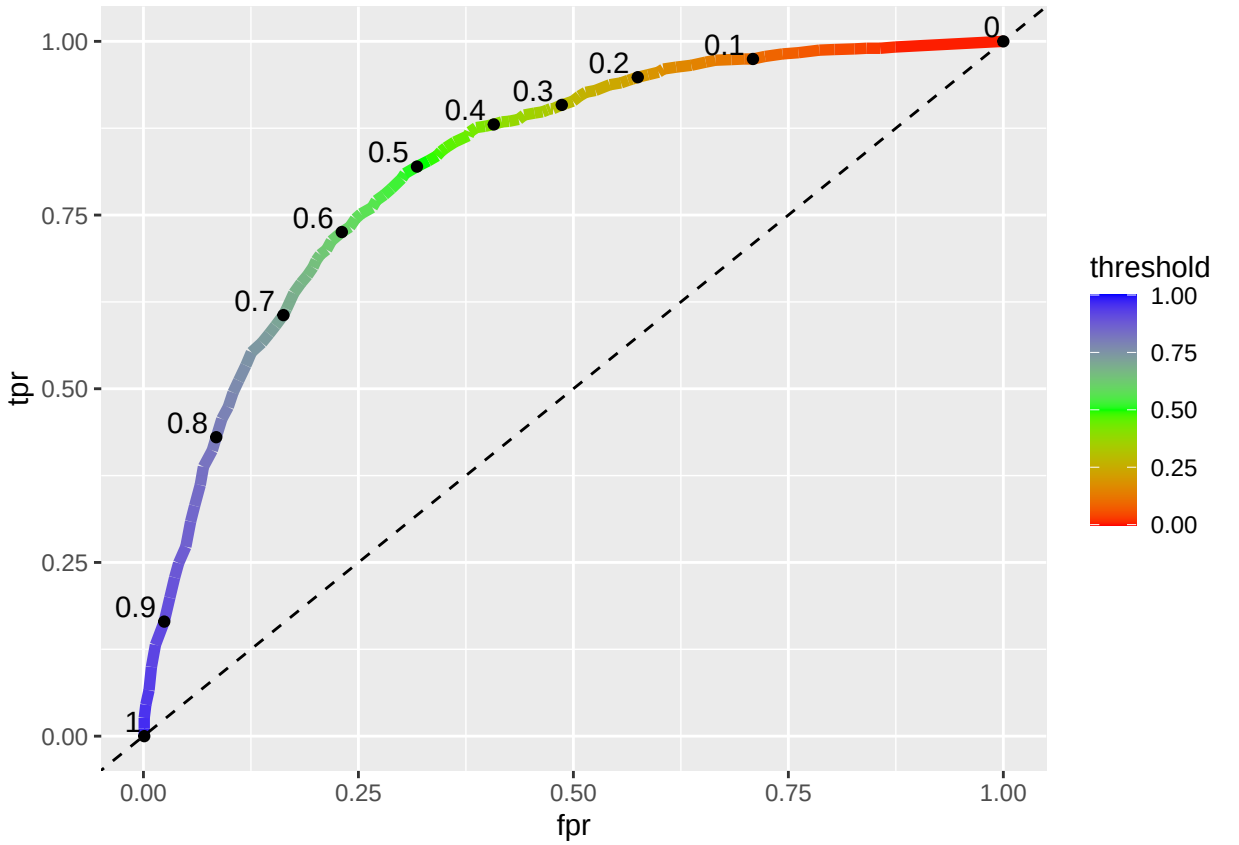
```
## [1] 0.1190698
```

```
tpr <- sum(over_threshold$lodgepole_pine==1)/sum(test_set$lodgepole_pine==1)
tpr
```

```
## [1] 0.5335145
```

Now, we can decrease the FPR to 0.1190698, but at the cost of also decreasing the TPR to 0.5335145. The ROC curve shows the FPR versus the TPR for different threshold values ranging from 1 to 0. Let's plot the ROC curve by calculating the FPR and TPR values for the thresholds $1, 0.99, 0.98, \ldots, 0$. To plot the ROC curve, we first set up a dataframe that will hold the FPR and TPR values for each threshold, as shown in the code below. Then, we use a 'for' loop to iterate over the threshold values.

```
roc_data <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data$threshold) {

  over_threshold <- test_set[test_set$p_prob >= i, ]

  fpr <- sum(over_threshold$lodgepole_pine==0)/sum(test_set$lodgepole_pine==0)
  roc_data[roc_data$threshold==i, "fpr"] <- fpr

  tpr <- sum(over_threshold$lodgepole_pine==1)/sum(test_set$lodgepole_pine==1)
  roc_data[roc_data$threshold==i, "tpr"] <- tpr

}

ggplot() +
  geom_line(data = roc_data, aes(x = fpr, y = tpr, color = threshold), size = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))
```



Ideally, we want the ROC curve to hug the top left-hand corner, which means we can achieve a high TPR for a low FPR.

AUC

To sum up the ROC curve with one number, we calculate the area under the ROC curve (AUC). This can be done using any R function that can calculate the area under a curve, given the 'x' and 'y' values that define the curve. Here, we use the 'auc' function from the 'MESS' library.

```
auc <- auc(x = roc_data$fpr, y = roc_data$tpr, type = "spline")
auc
```

```
## [1] 0.8229929
```

The largest AUC value we can get is 1, which corresponds to a perfect classifier. In practice, what defines a good AUC value depends on the application and/or problem domain.

Calibration Curves

The calibration curve assesses how well our predicted probabilities are calibrated. For example, let's look at all of the observations in the test set that have a predicted probability of positive class membership between 0.7 and 0.8.

```
in_interval <- test_set[test_set$p_prob >= 0.7  & test_set$p_prob <= 0.8, ]
```

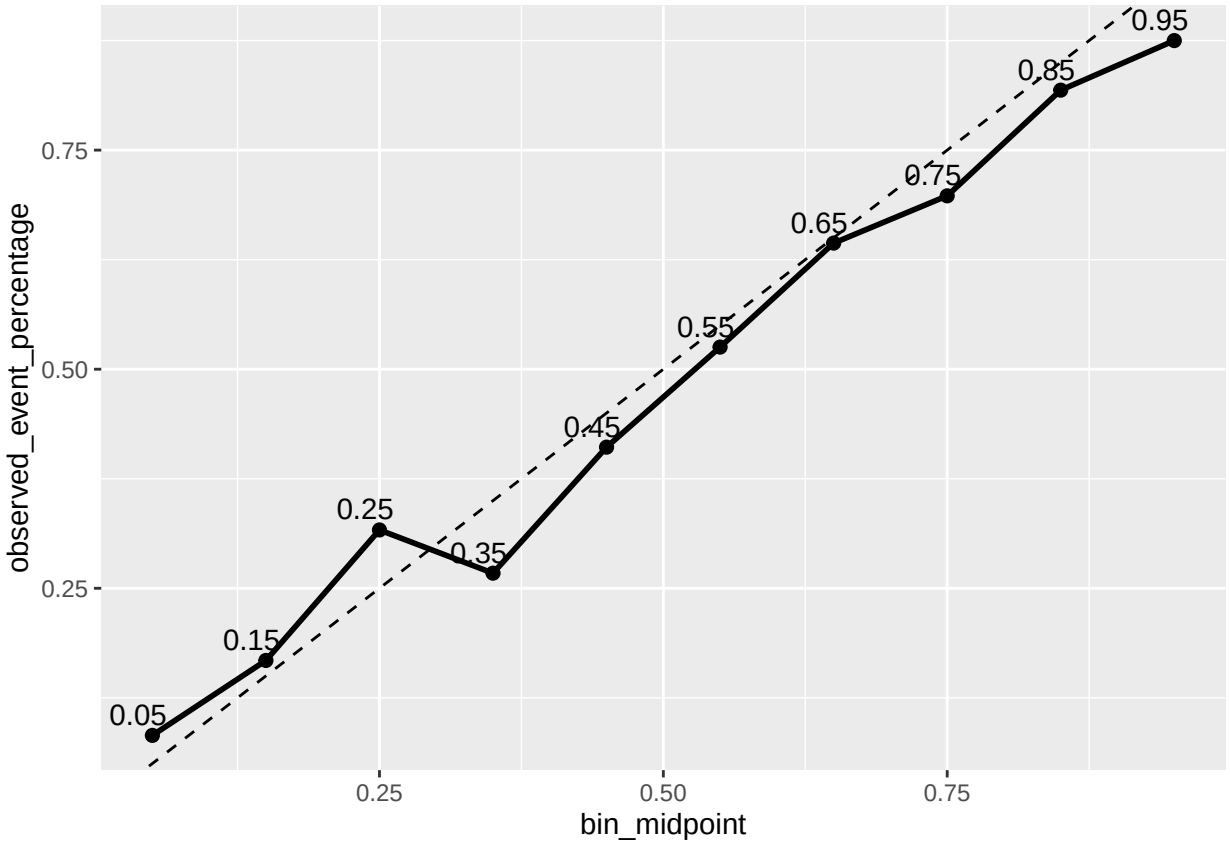
The dataframe 'in_interval' contains these observations. If the predicted probabilities are well-calibrated, we expect about 75% of these observations to belong to the positive class, since the midpoint of the interval (0.7, 0.8) is 0.75. The percentage of observations in 'in_interval' that belong to class 1 is called the observed event percentage for this interval, calculated below:

```
nrow(in_interval[in_interval$lodgepole_pine==1, ])/nrow(in_interval)
```

```
## [1] 0.6978417
```

Since the observed event percent 0.6978417 is close to 0.75, the probabilities in this interval seem to be well-calibrated. The calibration curve shows the observed event percentage for any set of intervals that partition the interval (0, 1), which is the range of possible predicted probability values. Here, we use the intervals (0, 0.1), (0.1, 0.2), (0.2, 0.3), (0.3, 0.4), (0.4, 0.5), (0.5, 0.6), (0.6, 0.7), (0.7, 0.8), (0.8, 0.9), and (0.9, 1).

```
calibration_data <- data.frame(bin_midpoint=seq(0.05,0.95,0.1),
                               observed_event_percentage=0)
for (i in seq(0.05,0.95,0.1)) {

  in_interval <- test_set[test_set$p_prob >= (i-0.05) & test_set$p_prob <= (i+0.05), ]
  oep <- nrow(in_interval[in_interval$lodgepole_pine==1, ])/nrow(in_interval)
  calibration_data[calibration_data$bin_midpoint==i, "observed_event_percentage"] <- oep

}

ggplot(data = calibration_data, aes(x = bin_midpoint, y = observed_event_percentage)) +
  geom_line(size = 1) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(size = 2) +
  geom_text(aes(label = bin_midpoint), hjust = 0.75, vjust = -0.5)
```



Ideally, we want the calibration curve to lie near the dashed diagonal line, which corresponds to a well-

calibrated classifier. Values that lie above the dashed diagonal line correspond to under-confident probabilities and values that lie below correspond to over-confident probabilities.

Exercises

1) In the ROC curve above, what is the TPR and FPR associated with the threshold value of 0.3?
2) In the calibration curve above, are the predicted probabilities in the interval (0.2, 0.3) under-confident or over-confident?
3) The 'AppliedPredictiveModeling' R package contains several datasets. One such dataset is the 'logisticCreditPredictions' dataframe, which contains the predictions and predicted probabilities for a credit dataset containing a binary target variable with the classes 'Good' and 'Bad'. The positive class is the 'Bad' class, since we are trying to identify customers with bad credit. The 'logisticCreditPredictions' dataframe has 4 columns: the columns 'Bad' and 'Good' contain the predicted probabilities of class membership, the column 'pred' contains the predicted class using the threshold 0.5, and the column 'obs' contains the actual class. Use the code below to plot an ROC curve and calibration curve for the predicted probabilities. To do this, fill in the question marks with the appropriate column names and values. Copy and paste the ROC curve and calibration curve.

```
# Hint: the column 'pred' is not needed.
# Hint: there are a total of 8 question marks.


library(AppliedPredictiveModeling)
data("logisticCreditPredictions")
lcp <- logisticCreditPredictions # only do this to shorten the name

#### ROC curve
roc_data <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)
for (i in roc_data$threshold) {

  over_threshold <- lcp[lcp$? >= i, ]

  fpr <- sum(over_threshold$obs==?)/sum(lcp$obs==?)
  roc_data[roc_data$threshold==i, "fpr"] <- fpr

  tpr <- sum(over_threshold$obs==?)/sum(lcp$obs==?)
  roc_data[roc_data$threshold==i, "tpr"] <- tpr

}

ggplot() +
  geom_line(data = roc_data, aes(x = fpr, y = tpr, color = threshold), size = 2) +
  scale_color_gradientn(colors = rainbow(3)) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(data = roc_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +
  geom_text(data = roc_data[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))

### calibration curve
calibration_data <- data.frame(bin_midpoint=seq(0.05,0.95,0.1),
                               observed_event_percentage=0)
for (i in seq(0.05,0.95,0.1)) {
```

```
  in_interval <- lcp[lcp$? >= (i-0.05) & lcp$? <= (i+0.05), ]
  temp <- nrow(in_interval[in_interval$obs==?, ])/nrow(in_interval)
  calibration_data[calibration_data$bin_midpoint==i, "observed_event_percentage"] <- temp

}

ggplot(data = calibration_data, aes(x = bin_midpoint, y = observed_event_percentage)) +
  geom_line(size = 1) +
  geom_abline(intercept = 0, slope = 1, lty = 2) +
  geom_point(size = 2) +
  geom_text(aes(label = bin_midpoint), hjust = 0.75, vjust = -0.5)
```