R Lab: Principal Component Analysis

In this lab, we use the ‘prcomp’ function, which is the R function for implementing principal component analysis (PCA). We first load the R libraries we’ll be using. Note, all of these libraries are for implementing the code from the previous labs: the ‘prcomp’ function is available in the base R installation, so no package is needed.

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

Principal Component Analysis

Before we implement PCA, we follow the same data pre-processing steps as in the previous labs.

```r
data <- read.csv("lab_7_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)
```

```r
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)))
```

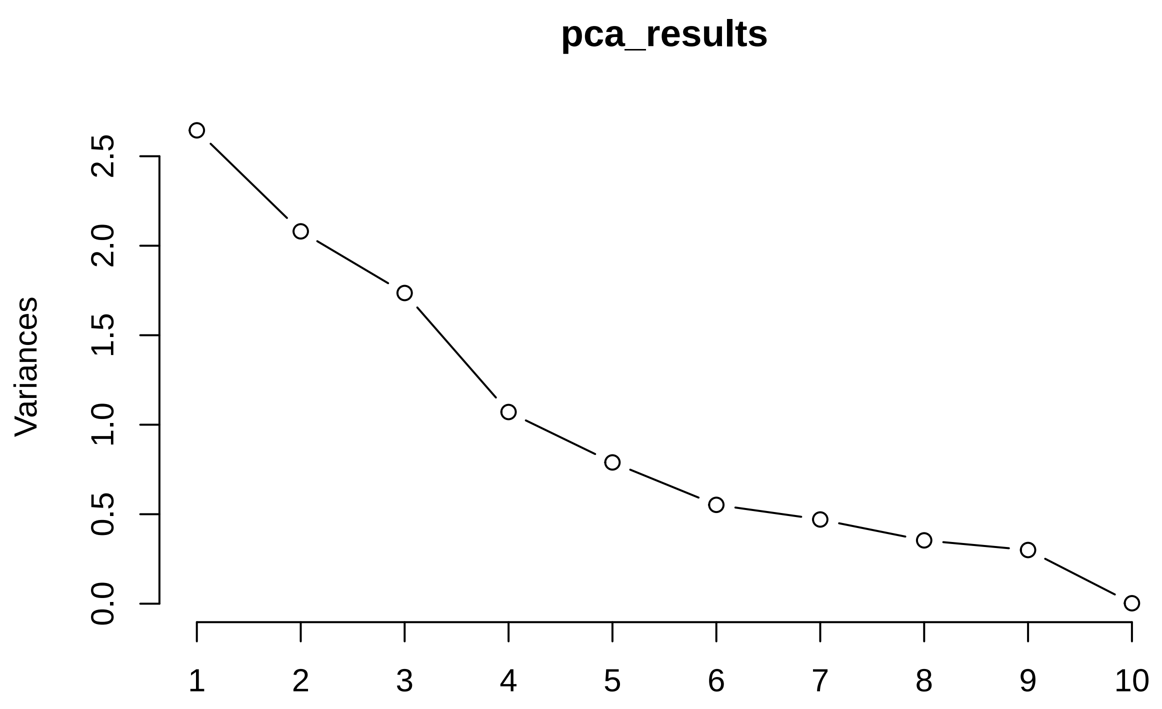
Note, as discussed in the live session, the scaling of the training and test sets above is critical for PCA. Now that the features are scaled, we implement PCA in R using the 'prcomp' function. The numerical features are contained in the first 10 columns of the data, so we run PCA only on the first 10 columns. The output is obtained using the 'summary' function.

```r
pca_results <- prcomp(training_features[, 1:10])
summary(pca_results)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     1.6262 1.4423 1.3175 1.0349 0.88842 0.74314 0.68596
## Proportion of Variance 0.2645 0.2080 0.1736 0.1071 0.07893 0.05523 0.04705
## Cumulative Proportion  0.2645 0.4725 0.6461 0.7531 0.83208 0.88731 0.93436
##                            PC8     PC9    PC10
## Standard deviation     0.59486 0.54765 0.05097
## Proportion of Variance 0.03539 0.02999 0.00026
## Cumulative Proportion  0.96975 0.99974 1.00000
```

The summary of the PCA displays several things. Most importantly, we can see the proportion of the variance and the cumulative proportion of the variance explained by each principal component (PC). In this example, the first PC explains about 26% of the variance. This output can be used to choose the number of PCs that explain enough of the variance. However, a scree plot is more convenient.

```r
screeplot(pca_results, type = "line")
```



In the scree plot, we look for an 'elbow' in the curve to help us choose the number of PCs that explain enough of the variance. However, as can be seen in the plot above, it is not always straightforward to identify the elbow; unsupervised learning is much more subjective than supervised learning and depends on the judgement of the analyst. In the plot above, the first 6 PCs seem to explain enough of the variance (almost 90%). Therefore, we can try reducing the number of numerical features from 10 to 6. The new 6 features are the projections of the original observations onto the first 6 PCs.

To obtain these projections, we multiply the original data (in matrix form) by the 'rotation' component of the 'pca_results' object. The operation '%*%' performs matrix multiplication, where the operands are both matrices.

```
training_rotated <- as.matrix(training_features[, 1:10]) %*% pca_results$rotation
```

The dataframe 'training_rotated' contains the projections of the observations in the training set onto the PCs. In particular, each column in 'training_rotated' contains the projection onto the corresponding PC: the first column in 'training_rotated' contains the projection of the observations onto the first PC, the second column in 'training_rotated' contains the projection of the observations onto the second PC, etc. After we project the observations onto the PCs, we take only the first 6 columns (since we're only using the first 6 PCs) and append the new features onto the training set to have all of the features in one dataset.

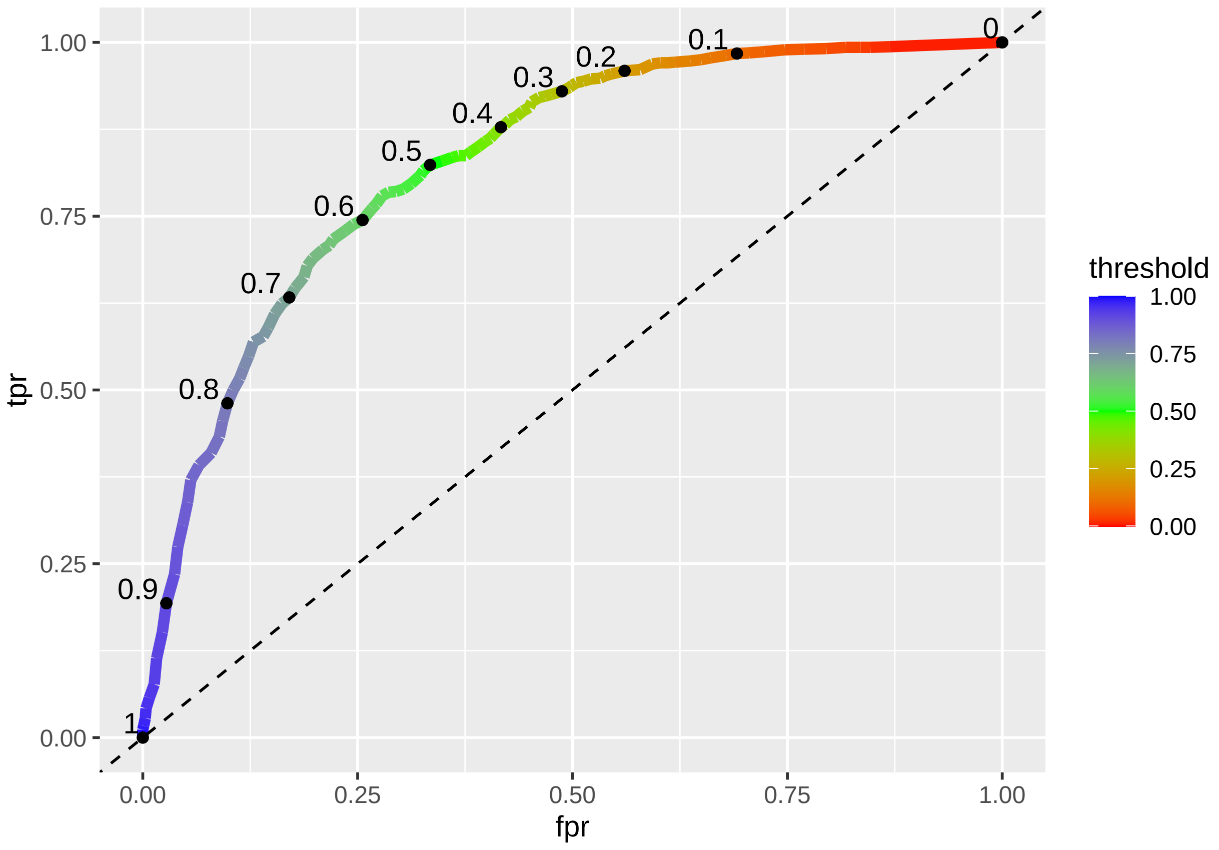
```
training_features <- cbind(training_features, training_rotated[, 1:6])
```

To project the observations in the test set onto the first 6 PCs, we perform the same operation as above on the test set. In particular, we use the 'rotation' component of the 'pca_results' object that was obtained using PCA on the training set (remember, everything is based on the training set!).

```
test_rotated <- as.matrix(test_features[, 1:10]) %*% pca_results$rotation
test_features <- cbind(test_features, test_rotated[, 1:6])
```

Now that the training and test sets each have the 6 new features obtained using PCA, we can try using these 6 new numerical features instead of the original 10. The code below is from the Random Forest Lab, with one small change: we use the index '-1*c(1:13)' for the 'x' parameter of the 'train' function, since we do not want to use the original 10 numerical features, the categorical features (wilderness_area and soil_type), or the target (lodgepole_pine) to train the model.

```
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)

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

##### ROC curve
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))
```



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

```
## [1] 0.8269805
```

Comparing the ROC curve and AUC above to the ROC curve and AUC for the original dense feed-forward neural network in Lab #5, we can see that both models are essentially indistinguishable, even though we used a reduced number of features in this lab. This is because the 6 new features (based on the first 6 PCs) account for almost 90% of the variation in the original 10 features

In this lab, we used the default method to implement PCA. However, several different methods exist and are also implemented in R. Look into the different methods, as well as other arguments, that are implemented in the 'prcomp' functions discussed above.

Exercises

Hint for all exercises: see the synchronous live session slides.

1) What is the main difference between unsupervised and supervised learning?
2) Is centering required for PCA? Is scaling required for PCA? Explain your answers.
3) After running the code above, run the following code to obtain the PCA feature loadings. Copy and paste the output. Which feature has the strongest influence on the first PC? Which feature has the weakest influence on the first PC?

```
pca_results$rotation
```