Week7_SimonsenHomework

Steven Simonsen

2024-10-11

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
library(reticulate)
library(tensorflow)
```

```
##
## Attaching package: 'tensorflow'

## The following object is masked from 'package:caret':
##
##     train
```

```r
library(keras3)
```

```
##
## Attaching package: 'keras3'

## The following objects are masked from 'package:tensorflow':
##
##     set_random_seed, shape
```

```r
library(MESS)

setwd("C:\\Users\\steve\\OneDrive\\Documents\\School\\DSE6211\\Week7")

data <- read.csv("lab_7_data.csv")

training_ind <- createDataPartition(data$lodgepole_pine,
                                    p = 0.75,
                                    list = FALSE,
                                    times = 1)
```

```r
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)
```

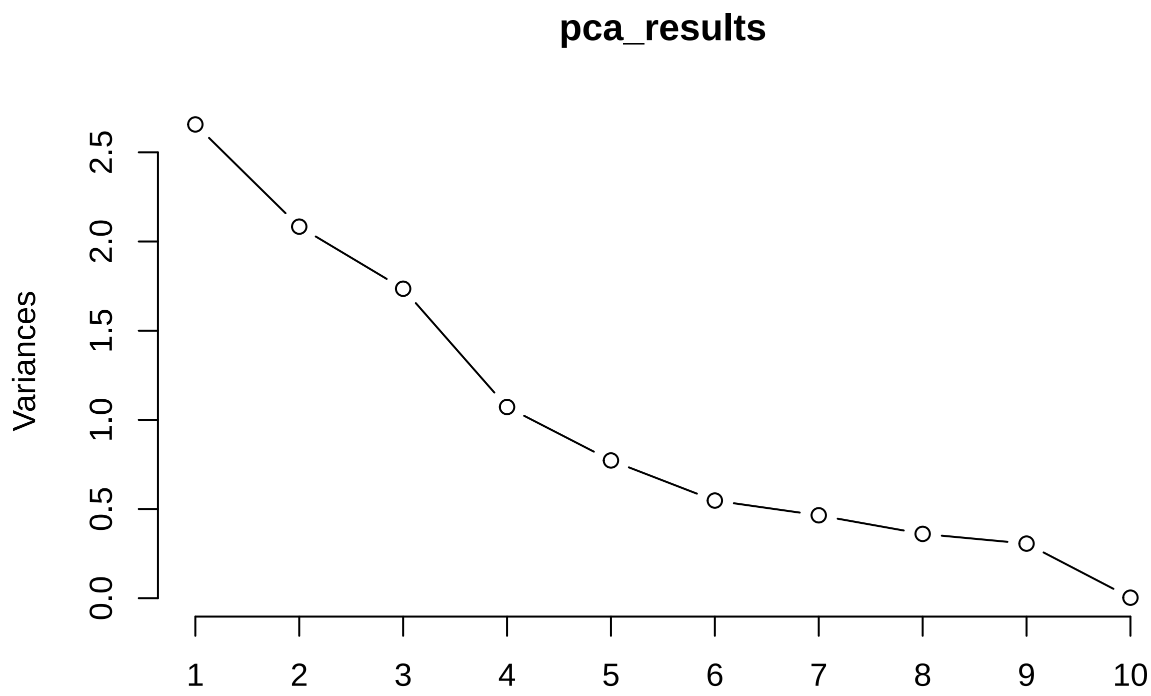
```
## Selecting by soil_type
```

```r
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))
```

```r
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)))

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

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     1.6298 1.4433 1.3172 1.0353 0.87873 0.73984 0.68168
## Proportion of Variance 0.2656 0.2083 0.1735 0.1072 0.07722 0.05474 0.04647
## Cumulative Proportion  0.2656 0.4739 0.6475 0.7547 0.83186 0.88660 0.93307
##                            PC8    PC9    PC10
## Standard deviation     0.60035 0.5532 0.05351
## Proportion of Variance 0.03604 0.0306 0.00029
## Cumulative Proportion  0.96911 0.9997 1.00000
```

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




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

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

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

test_features <- cbind(test_features, test_rotated[, 1:6])


use_virtualenv("my_tf_workspace")

model <- keras_model_sequential() %>%
  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,
```

```r
              epochs = 40, batch_size = 512,
              validation_split = 0.33)
```

```
## Epoch 1/40
## 9/9 - 1s - 139ms/step - accuracy: 0.5832 - loss: 0.6810 - val_accuracy: 0.4810 - val_loss: 0.7419
## Epoch 2/40
## 9/9 - 0s - 18ms/step - accuracy: 0.6894 - loss: 0.6003 - val_accuracy: 0.5185 - val_loss: 0.7187
## Epoch 3/40
## 9/9 - 0s - 9ms/step - accuracy: 0.7239 - loss: 0.5625 - val_accuracy: 0.5385 - val_loss: 0.7196
## Epoch 4/40
## 9/9 - 0s - 10ms/step - accuracy: 0.7518 - loss: 0.5353 - val_accuracy: 0.5579 - val_loss: 0.7139
## Epoch 5/40
## 9/9 - 0s - 21ms/step - accuracy: 0.7643 - loss: 0.5145 - val_accuracy: 0.5857 - val_loss: 0.7046
## Epoch 6/40
## 9/9 - 0s - 8ms/step - accuracy: 0.7721 - loss: 0.4971 - val_accuracy: 0.5918 - val_loss: 0.7070
## Epoch 7/40
## 9/9 - 0s - 20ms/step - accuracy: 0.7776 - loss: 0.4834 - val_accuracy: 0.6200 - val_loss: 0.6859
## Epoch 8/40
## 9/9 - 0s - 7ms/step - accuracy: 0.7879 - loss: 0.4722 - val_accuracy: 0.6223 - val_loss: 0.6915
## Epoch 9/40
## 9/9 - 0s - 8ms/step - accuracy: 0.7888 - loss: 0.4636 - val_accuracy: 0.6237 - val_loss: 0.6982
## Epoch 10/40
## 9/9 - 0s - 8ms/step - accuracy: 0.7885 - loss: 0.4570 - val_accuracy: 0.6469 - val_loss: 0.6674
## Epoch 11/40
## 9/9 - 0s - 10ms/step - accuracy: 0.7942 - loss: 0.4509 - val_accuracy: 0.6274 - val_loss: 0.6984
## Epoch 12/40
## 9/9 - 0s - 8ms/step - accuracy: 0.7945 - loss: 0.4468 - val_accuracy: 0.6492 - val_loss: 0.6814
## Epoch 13/40
## 9/9 - 0s - 10ms/step - accuracy: 0.7952 - loss: 0.4435 - val_accuracy: 0.6358 - val_loss: 0.7005
## Epoch 14/40
## 9/9 - 0s - 25ms/step - accuracy: 0.8027 - loss: 0.4391 - val_accuracy: 0.6450 - val_loss: 0.6889
## Epoch 15/40
## 9/9 - 0s - 9ms/step - accuracy: 0.8029 - loss: 0.4355 - val_accuracy: 0.6511 - val_loss: 0.6862
## Epoch 16/40
## 9/9 - 0s - 8ms/step - accuracy: 0.8038 - loss: 0.4334 - val_accuracy: 0.6520 - val_loss: 0.6941
## Epoch 17/40
## 9/9 - 0s - 7ms/step - accuracy: 0.8059 - loss: 0.4305 - val_accuracy: 0.6525 - val_loss: 0.6869
## Epoch 18/40
## 9/9 - 0s - 8ms/step - accuracy: 0.8054 - loss: 0.4277 - val_accuracy: 0.6409 - val_loss: 0.7050
## Epoch 19/40
## 9/9 - 0s - 8ms/step - accuracy: 0.8091 - loss: 0.4255 - val_accuracy: 0.6497 - val_loss: 0.6950
## Epoch 20/40
## 9/9 - 0s - 9ms/step - accuracy: 0.8095 - loss: 0.4225 - val_accuracy: 0.6399 - val_loss: 0.7045
## Epoch 21/40
## 9/9 - 0s - 10ms/step - accuracy: 0.8116 - loss: 0.4211 - val_accuracy: 0.6367 - val_loss: 0.7182
## Epoch 22/40
## 9/9 - 0s - 8ms/step - accuracy: 0.8150 - loss: 0.4189 - val_accuracy: 0.6399 - val_loss: 0.7085
## Epoch 23/40
## 9/9 - 0s - 10ms/step - accuracy: 0.8150 - loss: 0.4172 - val_accuracy: 0.6325 - val_loss: 0.7157
## Epoch 24/40
## 9/9 - 0s - 11ms/step - accuracy: 0.8155 - loss: 0.4145 - val_accuracy: 0.6381 - val_loss: 0.7023
## Epoch 25/40
## 9/9 - 0s - 8ms/step - accuracy: 0.8141 - loss: 0.4139 - val_accuracy: 0.6358 - val_loss: 0.7053
## Epoch 26/40
```

```
## 9/9 - 0s - 21ms/step - accuracy: 0.8169 - loss: 0.4113 - val_accuracy: 0.6214 - val_loss: 0.7324
## Epoch 27/40
## 9/9 - 0s - 11ms/step - accuracy: 0.8155 - loss: 0.4105 - val_accuracy: 0.6311 - val_loss: 0.7254
## Epoch 28/40
## 9/9 - 0s - 18ms/step - accuracy: 0.8196 - loss: 0.4088 - val_accuracy: 0.6247 - val_loss: 0.7279
## Epoch 29/40
## 9/9 - 0s - 7ms/step - accuracy: 0.8164 - loss: 0.4065 - val_accuracy: 0.6487 - val_loss: 0.6991
## Epoch 30/40
## 9/9 - 0s - 9ms/step - accuracy: 0.8191 - loss: 0.4059 - val_accuracy: 0.6409 - val_loss: 0.7151
## Epoch 31/40
## 9/9 - 0s - 7ms/step - accuracy: 0.8155 - loss: 0.4058 - val_accuracy: 0.6293 - val_loss: 0.7287
## Epoch 32/40
## 9/9 - 0s - 10ms/step - accuracy: 0.8226 - loss: 0.4033 - val_accuracy: 0.6348 - val_loss: 0.7215
## Epoch 33/40
## 9/9 - 0s - 9ms/step - accuracy: 0.8212 - loss: 0.4016 - val_accuracy: 0.6242 - val_loss: 0.7444
## Epoch 34/40
## 9/9 - 0s - 10ms/step - accuracy: 0.8242 - loss: 0.4004 - val_accuracy: 0.6247 - val_loss: 0.7406
## Epoch 35/40
## 9/9 - 0s - 10ms/step - accuracy: 0.8244 - loss: 0.3993 - val_accuracy: 0.6251 - val_loss: 0.7360
## Epoch 36/40
## 9/9 - 0s - 10ms/step - accuracy: 0.8262 - loss: 0.3976 - val_accuracy: 0.6270 - val_loss: 0.7353
## Epoch 37/40
## 9/9 - 0s - 9ms/step - accuracy: 0.8244 - loss: 0.3960 - val_accuracy: 0.6344 - val_loss: 0.7334
## Epoch 38/40
## 9/9 - 0s - 9ms/step - accuracy: 0.8274 - loss: 0.3958 - val_accuracy: 0.6348 - val_loss: 0.7316
## Epoch 39/40
## 9/9 - 0s - 9ms/step - accuracy: 0.8253 - loss: 0.3943 - val_accuracy: 0.6177 - val_loss: 0.7745
## Epoch 40/40
## 9/9 - 0s - 8ms/step - accuracy: 0.8258 - loss: 0.3932 - val_accuracy: 0.6330 - val_loss: 0.7406
```

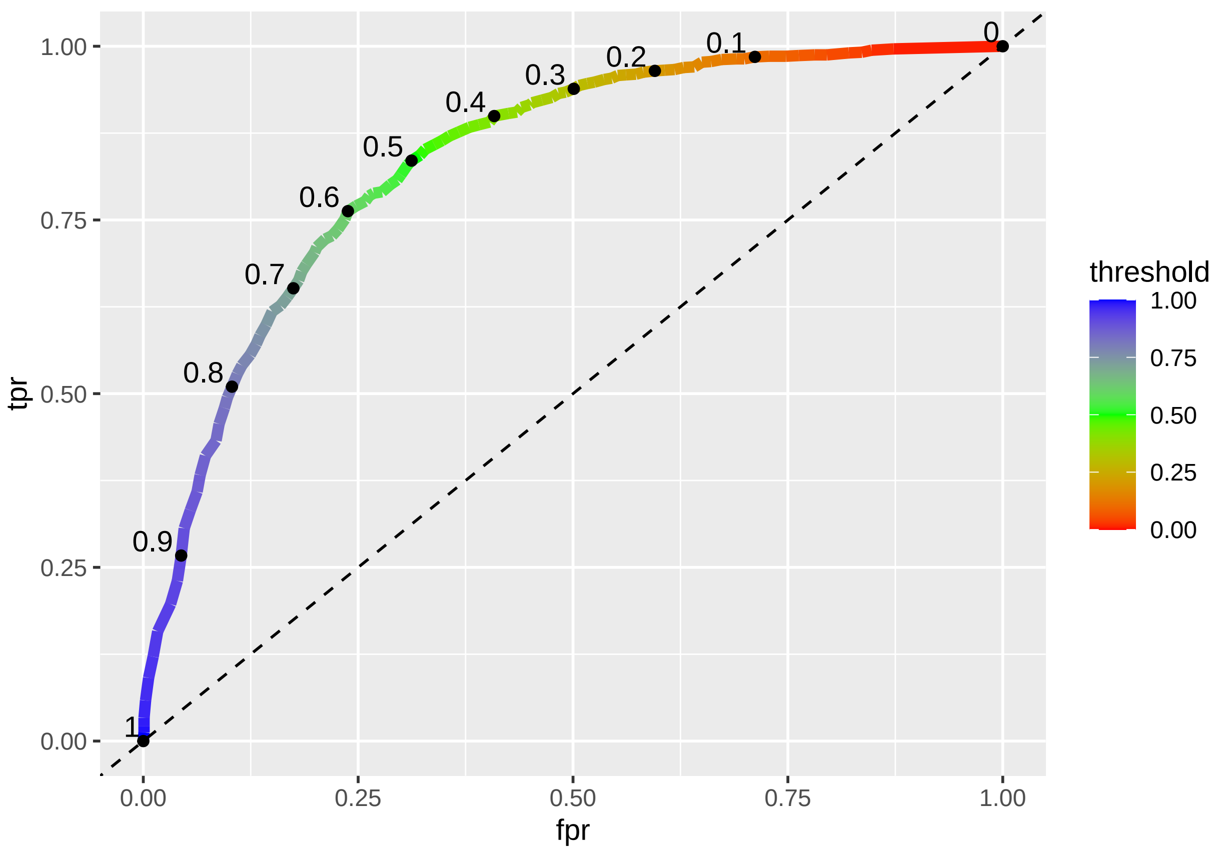
```r
predictions <- predict(model, test_features)
```

```
## 69/69 - 0s - 2ms/step
```

```r
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 =
```

```r
                                                            tpr)) +
  geom_text(data = roc_data[seq(1, 101, 10), ],
            aes(x = fpr, y = tpr, label = threshold, hjust = 1.2,
                vjust = -0.2))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




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

```
## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique
## 'x' values
```

```r
auc
```

```
## [1] 0.8352148
```

Exercises

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

1) What is the main difference between unsupervised and supervised learning?

In supervised learning, each observation is associated with a label. In unsupervised learning, there are no labels associated with the observations.

2) Is centering required for PCA? Is scaling required for PCA? Explain your answers.

Centering is always required for PCA since PCs are unit vectors originating from the origin. Scaling is almost always required to avoid PCA finding the features with the most variance. The exception is if numerical features are measured on similar scales and we want features with larger variance to have more imporatance.

3) After running 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
```

```
##               PC1         PC2         PC3          PC4         PC5         PC6
##  [1,]  0.13408619  0.35811179 -0.33839272  0.033709165 -0.63116833 -0.53751583
##  [2,]  0.48351213 -0.16505835  0.08552072  0.079881768 -0.03061640  0.03863554
##  [3,] -0.11363833 -0.52197760 -0.15333845  0.378800022 -0.16016438  0.03699555
##  [4,]  0.11426905  0.02682936 -0.64515588 -0.231499324  0.16433327 -0.01288564
##  [5,]  0.09053546 -0.21476876 -0.61233625 -0.143458159  0.20354084  0.24132647
##  [6,]  0.13536836  0.39338019 -0.13623518  0.518875575 -0.26320448  0.66744328
##  [7,] -0.44266273  0.36461557 -0.02225060 -0.287362173  0.02320579  0.19215802
##  [8,]  0.39567759  0.33073442  0.14046608 -0.357793846  0.14526955  0.18326804
##  [9,]  0.58541387 -0.04395050  0.11321176 -0.006996245  0.08044702 -0.08325563
## [10,] -0.02289386  0.35524111 -0.10899245  0.543089950  0.63969694 -0.35621193
##               PC7         PC8         PC9          PC10
##  [1,] -0.12505664  0.16000417 -0.09786284  0.0040664244
##  [2,] -0.64536842 -0.50527345 -0.22592870 -0.0011724550
##  [3,] -0.36922408  0.36326524  0.48502163 -0.1306454849
##  [4,]  0.12583264 -0.44971705  0.52072259  0.0014025903
##  [5,] -0.05986715  0.36382628 -0.56082145 -0.0003350351
##  [6,]  0.14090663 -0.06320352  0.02286573 -0.0024913693
##  [7,] -0.44342946 -0.01726032  0.01431296 -0.5945060184
##  [8,] -0.28866482  0.46051936  0.33632123  0.3508546584
##  [9,]  0.29575964  0.17583456  0.06581013 -0.7115949348
## [10,] -0.15757721  0.07233475 -0.01702224  0.0013460826
```

The feature with the strongest influence on the first PC is feature number 9 since this is the feature with the largest absolute value, and the feature with the weakest influence on the first PC is feature number 10 since this has the smallest absolute value.