Lab 1 Block 1 Assignment 1

Alejo Perez Gomez

15/11/2020

1

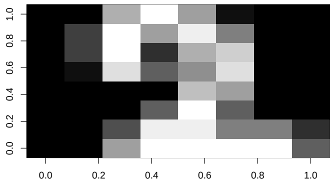
Import the data into R and divide it into training, validation and test sets (50%/25%/25%) by using the partitioning principle specified in the lecture slides.

```
digits <- read.csv(
  file = "C:/Users/alejo/Documents/GitHub_Repos/ML-labs/lab1/optdigits.csv")
digits[,ncol(digits)] <- data.frame(sapply(digits[,ncol(digits)],
                                        as.character),stringsAsFactors = TRUE)
names(digits) <- c(seq.int(ncol(digits)-1),"target")
```

Train/Valation/Test Division

```
set.seed(12345)

n=dim(digits)[1]
set.seed(12345)

id=sample(1:n, floor(n*0.5))
dfTraining=digits[id,]

id1=setdiff(1:n, id)
set.seed(12345)

id2=sample(id1, floor(n*0.25))
dfValidation=digits[id2,]

id3=setdiff(id1,id2)
dfTest=digits[id3,]
```

Plotting one observation



```
## [1] 2
```

2

Use training data to fit 30-nearest neighbor classifier with function kknn() and kernel=”rectangular” from package kknn and estimate - Confusion matrices for the training and test data (use table()) - Misclassification errors for the training and test data Comment on the quality of predictions for different digits and on the overall prediction quality.

Let’s fit the Model and predict results for Testing Set.

This will be our Confusion Matrix.

```
kknn_model <- kknn(dfTraining[["target"]] ~ ., dfTraining, dfTest, k = 30,
                   kernel = "rectangular")

fit <- fitted(kknn_model)
CM_test <- table(dfTest$target, fit)
CM_test
```

```
##    fit
##      0  1  2  3  4  5  6  7  8  9
##   0 98  0  0  0  0  0  0  0  0  0
##   1  0 92  3  0  0  0  0  0  0  2
##   2  0  0 93  1  0  0  0  0  1  0
##   3  0  0  0 95  0  0  0  2  1  0
##   4  1  0  0  0 89  0  1  5  1  3
##   5  0  1  0  0  0 81  0  0  0  5
##   6  0  0  0  0  0  0 94  0  0  0
##   7  0  2  0  0  0  0  0 92  1  0
##   8  0  3  0  1  0  0  1  0 86  0
##   9  0  0  0  3  0  0  0  2  0 96
```

```
accuracy_test <- (sum(diag(CM_test)))/sum(CM_test)
cat("Misclassification error in testing set is", 1 - accuracy_test)
```

```
## Misclassification error in testing set is 0.041841
```

Let’s fit the Model and predict results for Training Set. First we obtain the Confusion Matrix.

```
kknn_model <- kknn(dfTraining[["target"]] ~ ., dfTraining, dfTraining,
                   k = 30, kernel = "rectangular")
fit <- fitted(kknn_model)
CM_train <- table(dfTraining$target, fit)
CM_train
```

```
##    fit
##       0   1   2   3   4   5   6   7   8   9
##   0 177   0   0   0   1   0   0   0   0   0
##   1   0 174   9   0   0   0   1   0   1   3
##   2   0   0 171   0   0   0   0   1   1   0
##   3   0   0   0 198   0   1   0   1   0   0
##   4   0   1   0   0 168   0   1   6   1   2
##   5   0   0   0   0   0 186   0   2   0   9
##   6   0   0   0   0   0   0 200   0   0   0
##   7   0   1   0   1   0   0   0 193   0   0
##   8   0   7   0   1   0   0   1   0 196   0
##   9   0   3   0   2   2   0   0   2   3 184
```

```
accuracy_train <- (sum(diag(CM_train)))/sum(CM_train)
cat("Misclassification error in training set is", 1 - accuracy_train)
```

```
## Misclassification error in training set is 0.03349032
```

As seen in the results, misclassification test error surpasses training error which shows absence of underfitting. Classification behavior looks quite good since the error of misclassification doesn't exceed 0.05 in neither of the cases.

3

Find any 2 cases of digit "8" in the training data which were easiest to classify and 3 cases that were hardest to classify (i.e. having highest and lowest probabilities of the correct class). Reshape features for each of these cases as matrix 8x8 and visualize the corresponding digits (by using e.g. heatmap() function with parameters Colv=NA and Rowv=NA) and comment on whether these cases seem to be hard or easy to recognize visually.

First we get the probabilities of predictions, joining them with the actual category and the prediction made.

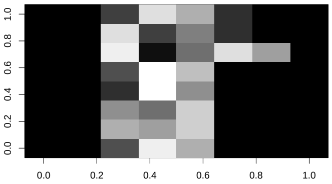
```
prob_df <- kknn_model$prob
max_prob <- colnames(prob_df)[apply(prob_df,1,which.max)]
train_probs <- data.frame(prob_df)
train_probs$actual <- dfTraining$target
train_probs$pred <- kknn_model$fitted.values
train_probs$max_prob <- max_prob
```

Let us subset the actual 8s and ordering by probabilities.

```
actual_8s <- train_probs[train_probs["actual"]==8,]
predicted_actual_8s <- actual_8s[actual_8s["pred"]==8,]
```

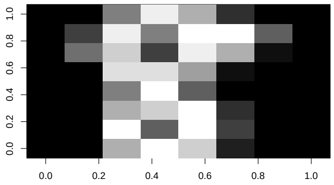
8 predicted correctly with high certainty and plots.

```
best_8_indexes <- as.numeric(
  row.names(predicted_actual_8s[order(-predicted_actual_8s[,9]),][1:2,]))

plot_observation(best_8_indexes[1], dfTraining)
```



```
## [1] 8
```

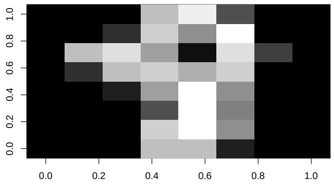
```
plot_observation(best_8_indexes[2], dfTraining)
```



```
## [1] 8
```

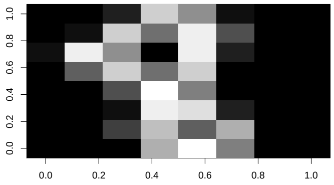
8 predicted correctly with low certainty and plots.

```
worst_8_indexes <- as.numeric(
  row.names(predicted_actual_8s[order(predicted_actual_8s[,9]),][1:3,]))
plot_observation(worst_8_indexes[1], dfTraining)
```



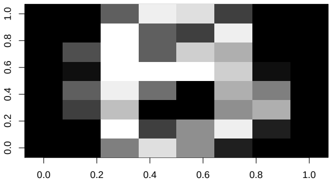
```
## [1] 8
```

```
plot_observation(worst_8_indexes[2], dfTraining)
```



```
## [1] 8
```

```
plot_observation(worst_8_indexes[3], dfTraining)
```



```
## [1] 8
```

Undoubtedly, the classified observations as 8s with less certainty are visually harder to tell apart from other numbers as the trace is pretty unclear. This doesn't happen to the correctly classified with high certainty.

4

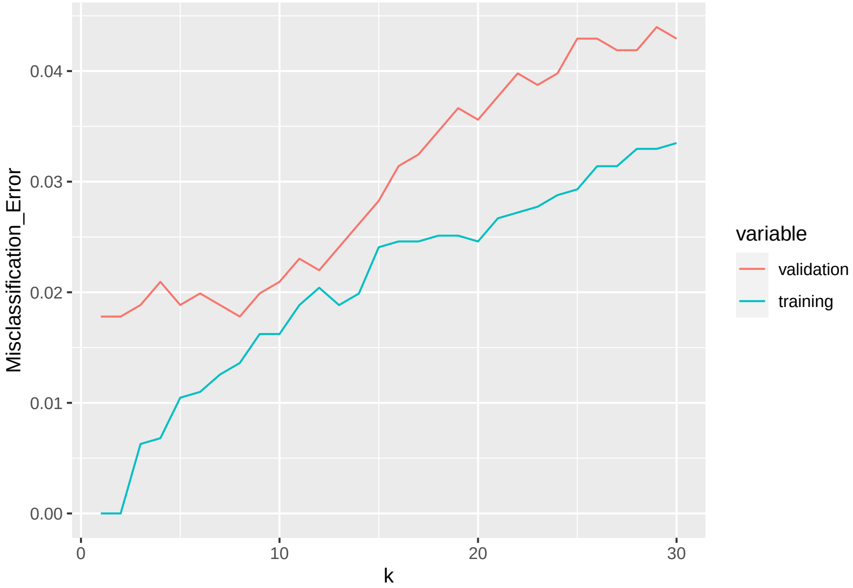
Fit a K-nearest neighbor classifiers to the training data for different values of $K = 1, 2, \ldots, 30$ and plot the dependence of the training and validation misclassification errors on the value of K (in the same plot). How does the model complexity change when K increases and how does it affect the training and validation errors? Report the optimal K according to this plot. Discuss this plot from the perspective of bias-variance tradeoff. Finally, estimate the test error for the model having the optimal K, compare it with the training and validation errors and make necessary conclusions regarding the model quality.

Let's fit a model for each k in the range.

```
KKNN_misclassification_training <- c()
KKNN_misclassification_validation <- c()

for (k in 1:30){

  # Fit for dfValidation and dfTraining

  KKNN_model_valid <- kknn(dfTraining[["target"]] ~ .,
                           dfTraining, dfValidation, k = k,
                           kernel = "rectangular")
  KKNN_model_train <- kknn(dfTraining[["target"]] ~ .,
                           dfTraining, dfTraining, k = k, kernel = "rectangular")

  # Calculation of Accuracies

  fit_train <- fitted(KKNN_model_train)
  CM_train <- table(dfTraining$target, fit_train)
  KKNN_misclassification_training[k] <- 1 - (sum(diag(CM_train)))/sum(CM_train)

  fit_valid <- fitted(KKNN_model_valid)
  CM_valid <- table(dfValidation$target, fit_valid)
  KKNN_misclassification_validation[k] <- 1 -(sum(diag(CM_valid)))/sum(CM_valid)


}
```

Plot of Accuracies.

```
metrics <- data.frame(KKNN_misclassification_validation,
                      KKNN_misclassification_training, 1:30)
names(metrics) <- c("validation", "training", "k")


metricsMelted <- melt(metrics, id.var='k')
names(metricsMelted)[3] <- "Misclassification_Error"
ggplot(metricsMelted, aes(x=k, y=Misclassification_Error,
                          col=variable)) + geom_line()
```



```
best_K <- (which(
  KKNN_misclassification_validation==min(KKNN_misclassification_validation)))
cat("The best performing K parameter are k =", as.character(best_K))
```

```
## The best performing K parameter are k = 1 2 8
```

The complexity of the model increases as K increases. For instance with $K = 1$ the error in training data is minimum resulting in minimum bias, whereas the variance is great as there are higher chances of getting error in testing set (generalizes worse, overfits). The training data is getting memorised at lowest K's, resulting in high variance. With higher K, training error increases (bias increases) but testing error decreases (decreasing variance) up to a point (optimal K, the sweet spot). Luckily, our optimal K (lowest error produced in validation set) coincides with a span where low variance and bias is attained. Before this spot, variance is decreasing and it starts increasing when it gets it passed. In regards to bias, the relationship with k is not that smooth, roughly it decreases until it reaches optimal K end then starts to increase again.

Testing of optimal K with Testing set. This is the Confusion Matrix for the optimal K in the Testing set.

```
kknn_model <- kknn(dfTraining[["target"]] ~ .,
                   dfTraining, dfTest, k = max(best_K), kernel = "rectangular")

fit <- fitted(kknn_model)
CM_test <- table(dfTest$target, fit)
CM_test
```

```
##    fit
##      0  1  2  3  4  5  6  7  8  9
##   0 97  0  0  0  0  0  1  0  0  0
##   1  0 93  3  0  0  0  0  0  0  1
##   2  0  1 93  0  0  0  0  1  0  0
##   3  0  0  0 95  0  0  0  2  1  0
##   4  1  0  0  0 92  0  1  3  0  3
##   5  0  0  0  0  0 85  0  0  0  2
##   6  0  0  0  0  0  0 94  0  0  0
##   7  0  1  0  0  0  0  0 92  1  1
##   8  0  3  0  0  0  0  0  0 88  0
##   9  0  0  0  3  1  0  0  0  0 97
```

```r
accuracy_test <- 1- (sum(diag(CM_test)))/sum(CM_test)
cat("Misclassification error is", accuracy_test)
```

```
## Misclassification error is 0.03138075
```

The error obtained in testing set classification is comparable to the error got among the best performing K at validation test classification. It leads us to thing the model is generalising properly.

5

Fit K-nearest neighbor classifiers to training data for different values of $K = 1, 2, \dots, 30$, compute the empirical risk for the validation data as cross-entropy (when computing log of probabilities add a small constant within log, e.g. 1e-15, to avoid numerical problems) and plot the dependence of the empirical risk on the value of K. What is the optimal K value here? Why might the cross-entropy be a more suitable choice of the empirical risk function than the misclassification error for this problem?

First we design our hot encode function to encode our target.

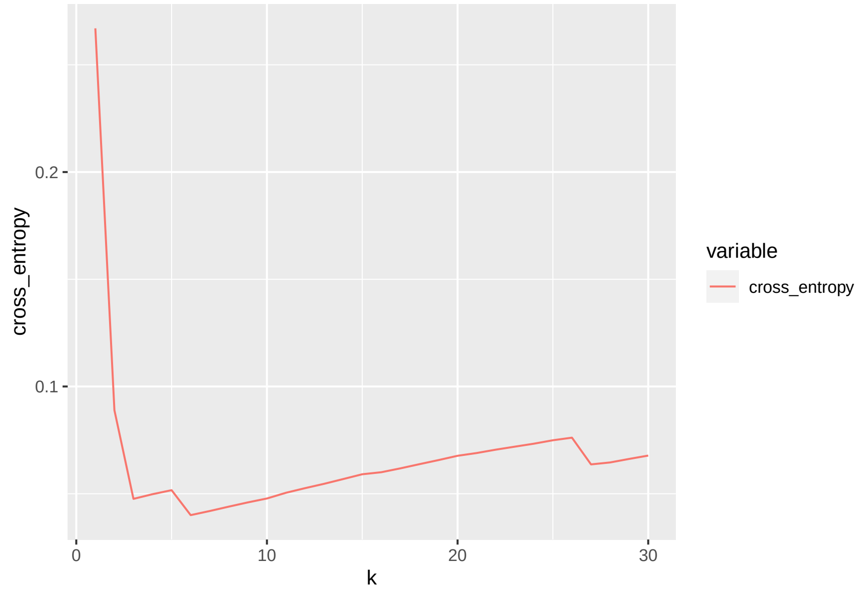
```r
hot_encode <- function(i){
  v <- rep(0,10)
  v[i+1] <- 1
  return(I(v))
}

empirical_risk <- c()
```

Let's fit a model for each k in the range. Then we will predict for the validation set and store the cross-entropy errors.

```r
for (k in 1:30){

  KKNN_model_valid <- kknn(dfTraining[["target"]] ~ .,
                           dfTraining, dfValidation, k = k,
                           kernel = "rectangular")

  # Extraction of probabilities, we build a df with the target hot encoded
  # to add a column of cross-entropy loss
  prob_df <- KKNN_model_valid$prob
  max_prob <- colnames(prob_df)[apply(prob_df,1,which.max)]
  valid_probs <- data.frame(prob_df)
  valid_probs$actual <- dfValidation$target
  valid_probs$pred <- KKNN_model_valid$fitted.values
  valid_probs$max_prob <- max_prob

  valid_probs$coded <- (lapply(as.numeric(valid_probs$actual )-1,  hot_encode))

  # column of cross-entropy loss
  for (row in 1:nrow(valid_probs)){
    valid_probs[row, "cross_entropy"] <- -sum(log(valid_probs[row, 1:10]+1e-15,
                                              base = 10)*valid_probs[[row,"coded"]] )
  }

  empirical_risk[k] <- mean(valid_probs$cross_entropy)

}
```

Plot of Empirical Risk for each K.

```
metrics <- data.frame(empirical_risk, 1:30)
names(metrics) <- c("cross_entropy", "k")

metricsMelted <- melt(metrics, id.var='k')

names(metricsMelted)[3] <- "cross_entropy"
ggplot(metricsMelted, aes(x=k, y=cross_entropy, col=variable)) + geom_line()
```



```
best_K <- which(empirical_risk==min(empirical_risk))
cat("The best performing K parameter is k =", as.character(best_K))
```

```
## The best performing K parameter is k = 6
```

As a conclusion, the choice of cross-entropy error is such a sensible procedure as there is a penalization involved as long as prediction probability towards the wrong class is given. Therefore, we are accounting for not only the actual target but also penalizing if uncertainty is present when predicting in some percentage for some classes.