Datamining & Statistical Learning

— Homework 6 —

Pablo González

Introduction.

For this assignment, we will be working with a filtered version of the zipcode data set. This data set has been previously filtered in order to only consider the classification problem between the numbers two and seven. Our data is composed of 7,291 observations and 257 variables coming from digitalized handwritten digits from envelopes of the U.S. postal service. For each row, the first variable contains the number that is represented in the 16x16 image and the remaining 256 variables represent the grayscale value of its respective pixel. It is also noteworthy to notice that no data is missing.

As it can be seen in Table 1, there is a small difference in the frequency each label appear in our training data. However, we do not consider this difference to be big enough to cause a significant bias in our classifiers.

Table 1: Frequency

Label	**N**	**Proportion**
0	1194	16.38 %
1	1005	13.78 %
2	731	10.03 %
3	658	9.02 %
4	652	8.94 %
5	556	7.63 %
6	664	9.11 %
7	645	8.85 %
8	542	7.43 %
9	644	8.83 %

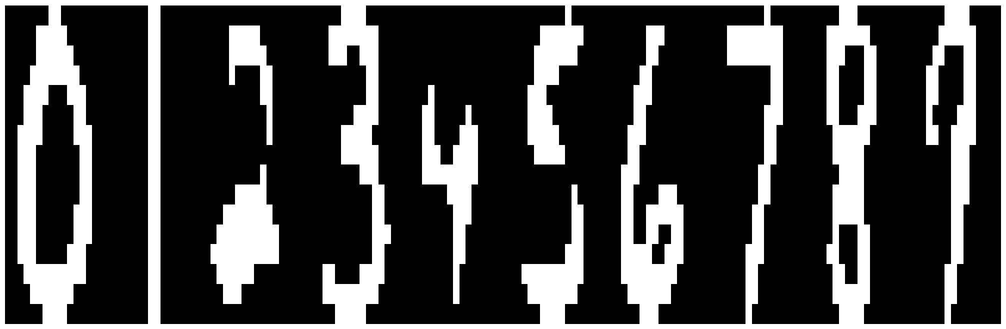
Additionally, Table 2 shows some basic summary statistics for the ten first predictors. As each one of them represent a certain pixel's grayscale value, it is possible to notice that, in general, their values range between -1.0 and 1.0.

Table 2: Descriptive statistics for the 10 first predictors

Stat	**V2**	**V3**	**V4**	**V5**	**V6**	**V7**	**V8**	**V9**	**V10**	**V11**
max	0.638	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
mean	-0.996	-0.981	-0.951	-0.888	-0.773	-0.610	-0.369	-0.046	-0.053	-0.285
min	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000	-1.000
var	0.003	0.023	0.060	0.130	0.253	0.395	0.513	0.590	0.567	0.558

To illustrate, if we calculate the mean value of each pixel for both of our labels and then render them, we will obtain the following ten numbers, which would represent how an average person using the U.S post service might write them respectively.

Figure 1: Mean values representation



Proposed Methodology

We would like to classify what number might a certain observation (image) represent. In order to do this, and only using the training data, we have trained six different classifiers structured as follows:

1. Linear discriminant analysis (LDA): This supervised method is commonly used for classification problems. Similar to principal components analysis (PCA), LDA is a dimensionality reduction technique, however, unlike PCA, LDA is also a supervised algorithm that allows for pattern classification by constructing a lower dimensional space that maximizes the variance between classes while minimizing the variance within each class.
2. Naive Bayes (NB): This probabilistic classifier assumes independence of the predictors and uses the Bayes rule to compute the conditional posterior probabilities of our response variable. For this analysis we use a gaussian naive bayes classifier, which implies that we also assume a conditional gaussian likelihood for the predictors.

3. Logistic Regression (LR): This classifier is an extension of the linear regression where the target variable is of categorical nature and is therefore predicted using the logit of the odds as the response variable. This method is commonly used for binary classification problems, as it makes no assumptions regarding the distribution of the target classes. However, it can become unstable when there is a clear separation between classes.

4. K-Nearest Neighbors (KNN): This is a supervised method that can be used to solve both regression problems (as in Homework 1) and classification problems such as predicting the corresponding written number. For this classifier, we estimated the best value for its K-parameter and checked our result by computing several values of K. Thus, when presenting our results, we will only report the resulting classifier for a K value of 3, however, Appendix A explains how we estimated the best hyper-parameter for this data set and shows the cross-validated testing error results for several values of K as well.

5. Random Forest (RF): Similar to the previous method, Random Forest can also be used both for regression and classification problems. In this case, this ensemble classifier operates by constructing 300 decision trees and taking the majority vote of these trees for prediction. As we know, ensemble learners generally outperform all their constituents, however, data characteristics can still affect their performance.

6. Boosting (BST): Boosting is also an ensemble learner. This supervised algorithm is characterized by being able to convert several weak learners to a strong one, thus allowing for bias an variance reduction. The optimal number of boosting iterations is estimated through cross-validations.

In the following section, we will analyze the results of the above presented models.

Analysis and Results

Table 3: Training errors by classifier

	NB	**LDA**	**LR**	**KNN**	**RF**	**Boosting**
Error	0.2277	0.0754	0.1358	0.0343	0.0233	0.0247

By analyzing this results, we can notice that the Random Forest classifier performs the best in our training data set with an error rate of 2.33%, closely followed by the Boosting classifier with a testing error of 2.47%. This is however expected, since ensemble algorithms are designed to have a smaller amount of bias and variance, thus being ideal for this kind of tasks and easily outperforming baseline classifiers such as the naive bayes classifier or the multinomial logistic regression.

Additionally, given that we have a large data set, we argue that our results are robust and do not require doing additional cross-validations, which are also computationally expensive given the complexity of the problem.

Finally, we test our chosen model in the testing data set, managing to correctly classify 94% of these images.

Conclusion

For this assignment, we worked with the zipcode data set in order to consider the classification problem between the numbers zero to nine. The problem consisted on being able to correctly distinguish the written number corresponding to a certain observation and label it accordingly by using 256 variables that represented the pixels from a 16x16 pixels image.

In order to solve this classification problem, we trained six different classifiers, including two ensemble learners, and calculated their testing errors, which we argue are robust given the size of our dataset, coming to the conclusion that the Boosting classifier has the best performance among the tested classifiers and the k=1 would be the best set-up for the KNN algorithm in order to solve this classification problem.

Finally, regarding the reason for which both ensemble learners outperformed the other classifieres, we proposed that this could be due to their ability to reduce both bias and variance by taking majority vote between several different weaker classifiers.

References

- This report was done using the “zipcode” data set. Available from: https://web.stanford.edu/~hastie/ElemStatLearn/data.html.
- Additional information regarding the employed data set can be found at: https://web.stanford.edu/~hastie/ElemStatLearn//datasets/zip.info.txt.
- Statistics were done using R 3.6.1 (R Core Team, 2019), the class (Ripley et al., 2020), tidyverse (Wickham et al., 2019), kableExtra (Zhao et al., 2020) and the knitr (Xie Y, 2020) packages.

Appendix

A. Estimation of the best K value.

With the aim of obtaining a robust estimator for the K hyper-parameter of our KNN classifier, we conducted 10 simulations for the first 10 values of K and focused on their mean testing error, noticing that, as expected from the mean numbers image, the value of K=1 seems to perform the best on this data set, given the presented statistics.

Table 4: Testing errors for the first 10 values of K

K	**min**	**mean**	**median**	**max**	**var**	**kurtosis**
1	0.01920	0.02565	0.02606	0.03292	2e-05	-1.40015
2	0.02332	0.03347	0.03361	0.03978	2e-05	-0.77705
3	0.02332	0.03032	0.02881	0.03978	3e-05	-1.35405
4	0.02743	0.03210	0.03224	0.04252	2e-05	-0.36481
5	0.01920	0.03306	0.03292	0.04115	4e-05	-0.10826
6	0.02469	0.03498	0.03429	0.04390	4e-05	-1.19183
7	0.02881	0.03621	0.03361	0.05213	5e-05	-0.36559
8	0.02743	0.03759	0.03498	0.05213	6e-05	-1.15997
9	0.02881	0.03992	0.04184	0.05350	8e-05	-1.69593
10	0.02881	0.03951	0.03978	0.05761	7e-05	-0.38003

B. R Code

```r
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE, fig.pos = 'H')
#Step 0: We import the necessary libraries.
if (!require('knitr')) install.packages('knitr'); library('knitr')
if (!require('kableExtra')) install.packages('kableExtra')
if (!require('class')) install.packages('class'); library('class')
if (!require('MASS')) install.packages('MASS'); library('MASS')
if (!require('e1071')) install.packages('e1071'); library('e1071')
if (!require('randomForest')) install.packages('randomForest')
if (!require('nnet')) install.packages('nnet'); library('nnet')
if (!require('xgboost')) install.packages('xgboost'); library('xgboost')
if (!require('rminer')) install.packages('rminer'); library('rminer')
if (!require('tidyverse')) install.packages('tidyverse'); library('tidyverse')
library('kableExtra'); library('randomForest')
select <- dplyr::select # To avoid mass & dplyr problem

shuffle <- function(data){

  row.names(data) <- NULL
  n = dim(data)[1]
  n1 = round(n/10)
  idx <- sort(sample(1:n, n1))
  x_train <- data[-idx,-1]
  y_train <- data[-idx,1]
  x_test <- data[idx,-1]
  y_test <- data[idx,1]

  return( list(x_train, y_train, x_test, y_test) )
}

get_errors <- function(data){
  temp <- shuffle(data)
  x_train = temp[[1]]; y_train = temp[[2]];
```

```r
x_test = temp[[3]]; y_test = temp[[4]];


# Naive Bayes
mod1 <- naiveBayes( x_train, y_train)
pred1 <- predict(mod1, x_test)
te1 <- mean( pred1 != y_test)


# LDA
mod2 <- lda(y_train ~., data = x_train)
pred2 <- predict(mod2,x_test)$class
te2 <- mean(pred2 != y_test)


# Logistic
mod3 <- nnet::multinom(y_train ~ ., data=x_train, maxit=400, MaxNWts = 4000)
pred3 <- predict(mod3, x_test)
te3 <- mean(pred3 != y_test)


# KNN
pred4 <- knn(x_train, x_test, y_train, k=1)
te4 <- mean(pred4 != y_test)


# Random Forest
mod5 <- randomForest(y_train ~ ., data=x_train,
                     ntree=500, boos=T, importance = 1)
pred5 <- predict(mod5, newdata = x_test)
te5 <- mean(pred5 != y_test)



# Boosting
tmptrain <- x_train %>% as.matrix() %>% xgb.DMatrix(data = ., label = y_train)
tmptest<- x_test %>% as.matrix() %>% xgb.DMatrix(data = ., label = y_test)


params <- list(booster = "gbtree", objective = "multi:softmax",
               num_class = 11, eval_metric = "mlogloss")
```

```r
  mod6 <- xgb.train(params=params, data=tmptrain, nrounds=500, nthreads=1,
    early_stopping_rounds=10, verbose=0,
    watchlist=list(val1=tmptrain,val2=tmptest) )

  pred6 = predict(mod6,tmptest,reshape=T) - 1
  te6 <- mean(pred6 != y_test)

  return(  data.frame(t(c(te1,te2,te3,te4,te5,te6))) )

}

knnerrors <- function(data, k = 1:5, reps = 10, seed = 7406){
  kvalues = c()
  for (i in k){
    set.seed(seed)
    aux = c()
    for (j in 1:reps){
      temp <- shuffle(data)
      x_train = temp[[1]]; y_train = temp[[2]];
      x_test = temp[[3]]; y_test = temp[[4]]

      pred <- knn(x_train, x_test, y_train, k=i)
      aux <- c(aux, mean(pred != y_test))
    }
    row <- data.frame(aux) %>%
      summarise_all(list(~min(., na.rm = T), ~mean(., na.rm = T),
                         ~median(., na.rm = T), ~max(., na.rm = T),
                         ~var(., na.rm = T), ~kurtosis(., na.rm = T)))
    kvalues <- rbind(kvalues, row)
  }
  kvalues = cbind(data.frame(K = k),data.frame(kvalues))
  return( data.frame(kvalues) )
}
```

```r
#Step 1: We import the datasets
zip <- read.table(file="zip.train.csv", sep = ",") %>%
  mutate(V1 =  as.factor(V1))
test <- read.table(file="zip.test.csv", sep = ",") %>%
  mutate(V1 =  as.factor(V1))
dim(zip) #The dataset contains 7291 observations with 257 variables.
sum(is.na(zip)) #There is no missing data
zip %>% group_by(V1) %>% summarise(n = n()) %>%
  mutate(freq = paste(round(100*n/sum(n),2), "%")) %>%
  kable(caption = "Frequency", align = "c",
        col.names = c("Label","N","Proportion")) %>%
  kable_styling("striped", full_width = F,
                latex_options = c("hold_position")) %>%
  row_spec (row = 0, bold = T)

header <- c("NB", "LDA","LR", "KNN", "RF", "Boosting")
zip %>% select(2:11) %>%
  summarise_all(list(~min(., na.rm = T), ~mean(., na.rm = T),
                     ~max(., na.rm = T), ~var(., na.rm = T))) %>%
  gather(variable, value) %>%
  separate(variable, c("var", "Stat"), sep = "\\_") %>%
  spread(var, value) %>% select(Stat, V2,V3,V4,V5,V6,V7,V8,V9,V10,V11) %>%
  kable(caption = "Descriptive statistics for the 10 first predictors",
        digits = 3, align = "l") %>%
  kable_styling("striped", full_width = T,
                latex_options = c("hold_position")) %>%
  row_spec (row = 0, bold = T)

meannumbers <- zip %>% group_by(V1) %>% summarise_all(~mean(.)) %>% select(-V1)

images = c()

for (i in 1:10){
img <- t(matrix(data.matrix( meannumbers[i,] ), byrow=TRUE,16,16)[16:1,])
images <- rbind(images, img)
}


image(images,col=gray(0:1),axes=FALSE)
title(main = "Figure 1: Mean values representation", font.main = 3)


set.seed(7406)
te <- get_errors(zip)
row.names(te) <- "Error"
colnames(te) <- header
te %>%  kable(caption = "Training errors by classifier",
              digits = 4, align = "l") %>%
  kable_styling("striped", full_width = T,
  latex_options = c("hold_position")) %>% row_spec (row = 0, bold = T)
selected <- randomForest(V1~., data = zip, ntree=500, importance = TRUE, boos=T)
pred <- predict(selected, newdata = test)
fe <- mean(pred != test[,1])
knnerrors(zip, k = 1:10, reps = 10, seed = 6047) %>%
  kable(caption = "Testing errors for the first 10 values of K",
    digits = 5, align = "l") %>% kable_styling("striped", full_width = T,
    latex_options = c("HOLD_position")) %>% row_spec (row = 0, bold = T)
```