732A99 - Lab 1, block 1 Group L4

Keshav Padiyar, Jacob Welander & Henrik Olofsson

18 November, 2020

Contents

statement of contribution

Keshav Padiyar: Assignment 1

Jacob Welander: Assignment 2

Henrik Olofsson: Assignment 3

Assignment 1. OverviewHandwritten Digit Recognition with K - Nearest Neighbour (knn) Classification

Dataset

The dataset contains information about normalized bitmaps of handwritten digits from a pre-printed form from a total of 43 people. The data were first derived as 32x32 bitmaps which were then divided into non-overlapping blocks of 4x4 and the number on pixels are counted in each block. This has generated the resulting image of size 8x8 where each element is an integer in the range 0 to 16. Accordingly, each row in the data-set is a sequence corresponding 8x8 matrix, and the last element contains the actual values of the digits i.e. 0 to 9.

Objective of the experiment

Using **knn** algorithm, considering the pixel values of each digit classify them into their respective classes (0-9). Identifying the optimal k value such that the error of classification is minimized.

Libraries used:

a. kknn
b. dplyr
c. reshape2
d. ggplot2

Methods Section:

Below are the methods written to achieve the objective:

a. **accuracy**: Takes confusion matrix as input, and calculates the accuracy (%) by considering the diagonal elements using the formula

$$accuracy = \frac{sum(diag(x))}{sum(rowSums(x)))} * 100,\ x = Confusion\ Matrix$$

b. **join_data_prob**: Method to map the cluster probabilities with the actual data and returns a data.frame.

c. **cross_entropy**: Method to calculate the Cross Entropy in the data sets and returns the log probabilities. Detailed description for this approach is given in section 5 below.

d. **func_compute**: This method is the computation engine, which takes the model object and target variable (vector) as input and returns a list having following values.

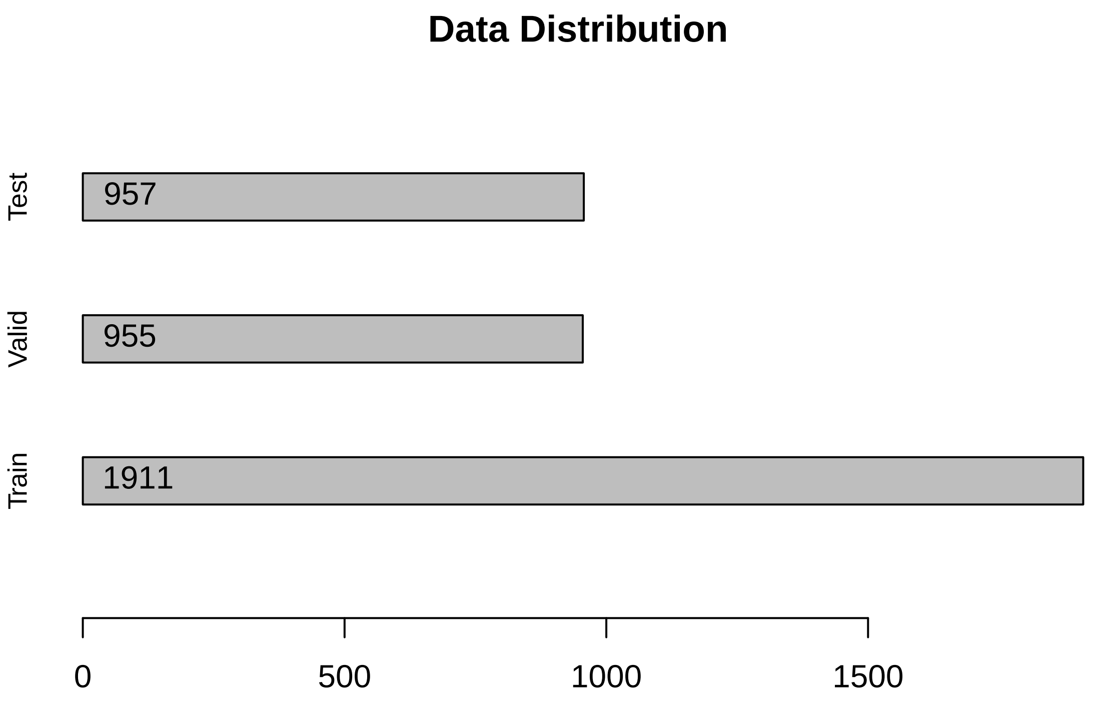
```
i. fitted values

ii. confusion matrix using table() function

iii. model accuracy

iv. miscalssification error, (100- accuracy)
```

```
  v. probabilities matrix

 vi. data.frame of probabilities matrix mappd with fitted
      and actual values

 vii. log probabilites of each pixel

viii. corss entropy value
```

1.

Importing the data from *optdigits.csv*. As per the *Holdout Method* dividing input data into training (50%), validation (25%) and testing (25%) data sets.

a. Training set is used for fitting the model,
b. validation set is used to validate the model for different values of k and choose the best model which has minimum error and hence low risk.
c. Finally, the test data is used to performing a test on the model and verify the model performance on this new data.



2.

kknn() package is used to implement the k nearest neighbor classification. The kknn method in the package, for each row of the train/test/valid set, the k nearest training set vectors (according to Minkowski distance) are found, and the classification is done via the maximum of summed kernel densities. In addition even ordinal and continuous variables can be predicted.

In this experiment, following parameters of kknn method are set:

a. formula: A formula object, i.e target variable (actual digits) given all pixel values. *target_variable* ~ .

b. train: Training data set.

c. test: Used Training data set while fitting the model, used validate for the validation and finally replaced with Test data for verifying the model performance.

d. kernel: "*rectangular*" kernel is used, which is standard un-weighted knn. kernel functions are used to weight the neighbors according to their distances.

e. k: Number of neighbors considered.

f. scale: Scale's the variable to have equal standard deviation.

```r
## 2
## Applying KKNN Algorithm on Training data set
cl_train <- kknn(as.factor(V65) ~ .,
                 train, test = train,
                 k = 30, kernel = "rectangular"
                 )

dt_cl_train = func_compute(cl_train, train_target)

## Applying KKNN Algorithm on TEST data set
cl_test <- kknn(as.factor(V65) ~ .,
                train, test = test,
                k = 30,
                kernel = "rectangular"
                )

dt_cl_test = func_compute(cl_test, test_target)
```

a. Confusion Matrix (using *table()*) of Training Data.

```
Confusion Matrix of Training Data:

      data_fit
target   0   1   2   3   4   5   6   7   8   9
     0 202   0   0   0   0   0   0   0   0   0
     1   0 179  11   0   0   0   0   1   1   3
     2   0   1 190   0   0   0   0   1   0   0
     3   0   0   0 185   0   1   0   1   0   1
     4   1   3   0   0 159   0   0   7   1   4
     5   0   0   0   1   0 171   0   1   0   8
     6   0   2   0   0   0   0 190   0   0   0
     7   0   3   0   0   0   0   0 178   1   0
     8   0  10   0   2   0   0   2   0 188   2
     9   1   3   0   5   2   0   0   3   3 183
```

b. Miscalssification Error for the Training and Test Data.

$$misclassification\ error = 100 - accuracy$$

```
Model with k = 30
```

```
Accuracy of model with Training Data:  95.49974 %

Misclassification Error in model with Training Data:  4.500262 %

Accuracy of model with Test Data:  94.67085 %

Misclassification Error in model with Test Data:  5.329154 %
```

Comment on the quality of predictions for different digits and on the overall prediction quality

Observing the Confusion Matrix and accuracy of the training, 95% 0f the data points got classified appropriately into respective classes. On the other hand, prediction accuracy decreased with the test data. We need to find the optimal value of k for more accuracy.

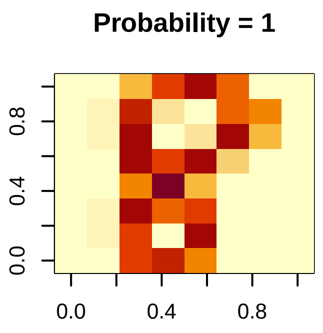
3.

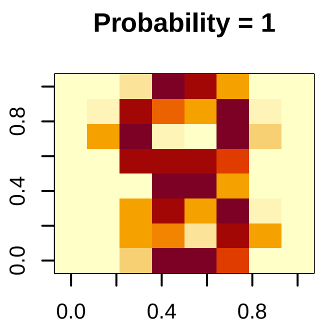
Reshaping the digit “8” for:

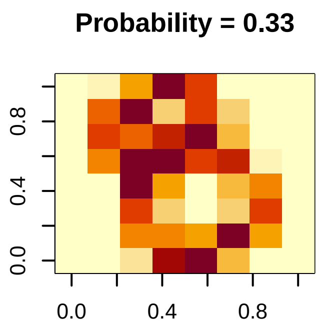
a. 2 cases where the training data were easiest to classify i.e rows with highest probabilities
b. 3 cases where the training data were hard to classify i.e rows with lowest probabilities.

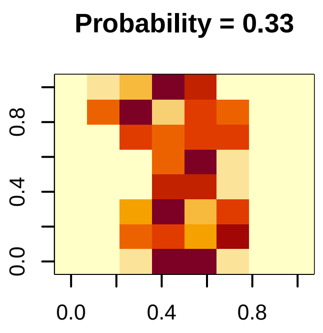
Used *Image()* function to visualize the pixel values.

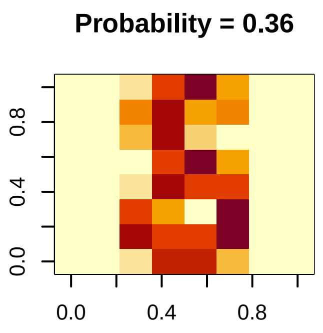
Referring to below visualization, pixels of 8 which got classified with highest probabilities are easy to recognize visually. On the otherhand the pixels that were classified with the lowest probabilities were hard to identify as they recemble some other digit.











4.

Fitting the model on training and validation data for k values ranging from 1 to 30.

4.1 How does the model complexity change when K increases and how does it affect the training and validation errors? As the k value increases, number of neighbors also increases hence model complexity increases.

In addition, beyond the optimal value of k, both train and validation errors are increasing.

4.2 The optimal k according to this plot: 3 or 4

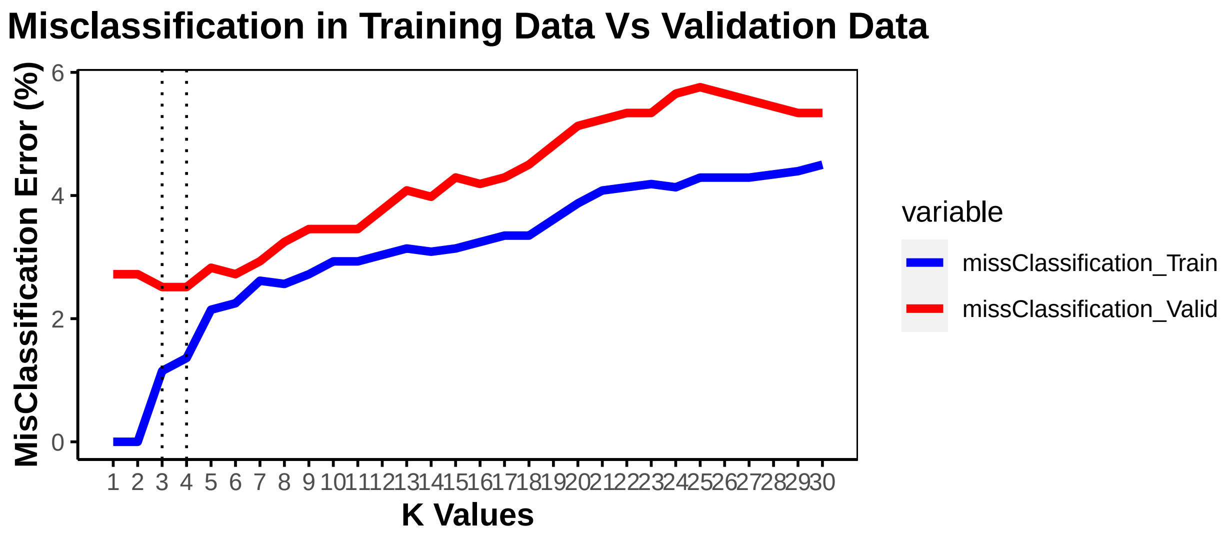
4.3 Discuss this plot from the perspective of bias-variance tradeoff

Referring to the below graph, for initial value of k, the training data gives 0% error, but could identify around 2% error in the validation data. It is due to overfit (bias is low and variance is high) on the training data for lower k value (k = 1 & 2).

As k value increases ($2 < k \leq 4$), gradual decrease in error in validation data is observed, but this is upto an optimal k. This shows that for those k values the bias is low and variance is decreasing.

At k = 3 & 4 there is a drop in the error of validation data and that k value has the lowest error. And it can be inferred that, at k = 3&4 bias and variances are optimal hence the error is lowest.

For k>4, both training and validation error rates eventually start increasing again, this is due to the underfitting (high bias).



4.4 Estimate the test error for the model having the optimal k

Obtaining the Accuracy and Misclassification Error for k = 4, using Training, Validation and Test data.

```
Model with K =  4

Accuracy of  model with Training Data:  98.63946 %

Misclassification Error in model with Training Data:  1.360544 %

Accuracy of Model with Validation Data:  97.48691 %

Misclassification Error in model with  validation Data:  2.513089 %
```

```
Accuracy of model with Test Data:  97.49216 %
```

```
Misclassification Error in model with Test Data:  2.507837 %
```

4.5 Compare test error with the training and validation errors and make necessary conclusions regarding the model quality

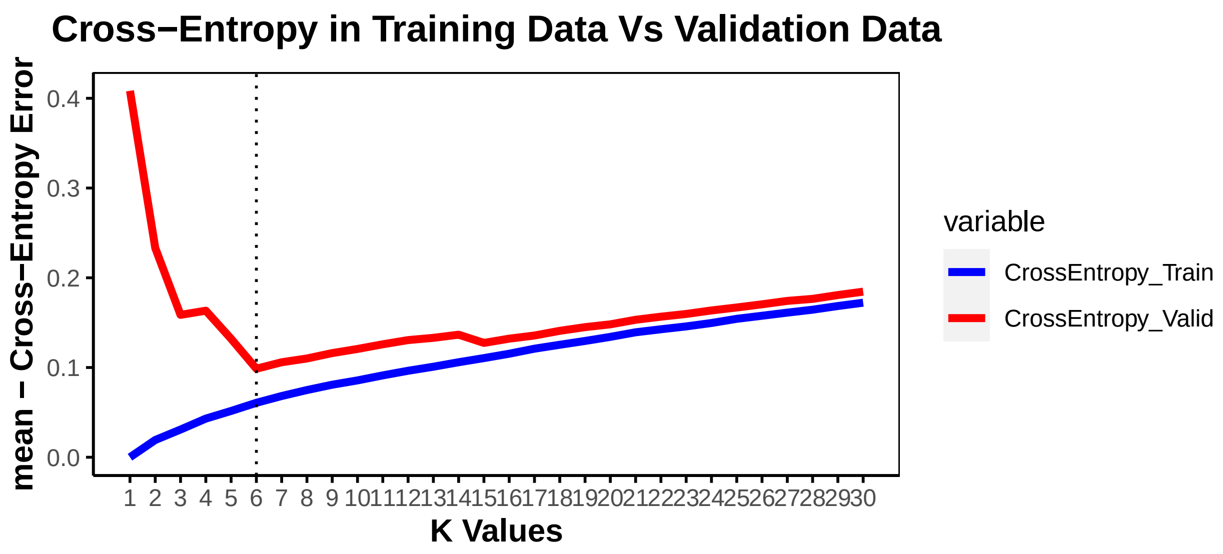
Looking at above missclassification error values, its inferred that for k = 4
the errors are lowest and improved the model performance.

5.

Cross Entropy measures the relationship between the estimated distribution and the actual distribution of the target variable.

Cross Entropy is calculated using the formula

$$CrossEntropy = -\sum_{i=1}^{n}\sum_{j=i}^{k} I(y_i = j)\log\left(\hat{p}_j(x_i) + 1e^{-15}\right)$$



5.1 What is the optimal k value here?

The above graph illustrates that, for k = 6 validation model shows the lowest error. Hence it can be inferred that, optimal value for k is 6.

5.2 Why might the cross-entropy be a more suitable choice of the empirical risk function than the misclassification error for this problem?

Cross entropy method considers the probability with which each data point has been classified into any of the classes, as a result of this the error is calculated from more granular level.

For example: considering the classification of digit 8. There are evidences where pixels are classified as digit 8 with the uncertainty (probability) = 0.333. Even with such a low uncertainty data points get classified as a result of majority voting. Cross Entropy penalizes such data points based on in the probability. Where as misclassification just compares the fitted value against the true value, hence cross entropy method gives more accurate result.

```
Model with K =  6
```

```
Cross Entropy Error in model with Training Data:  0.06067136

Cross Entropy Error in Model with Validation Data:  0.09866904
```

Assignenment 2. Ridge regression and model selection

1

Assuming that motor_UPDRS is normally distributed and can be modeled by Ridge regression of the voice characteristics, write down the probabilistic model as a Bayesian model.

A probabilistic Bayesian model where $\boldsymbol{y}$ and $\boldsymbol{w}$ is normally distributed as:

$$y \sim N\left(\boldsymbol{y} \mid \boldsymbol{X}\boldsymbol{w}, \sigma^2 I\right)$$
$$w \sim N\left(0, \tfrac{\sigma^2}{\lambda} I\right)$$

The function in later exercises should optimize $\boldsymbol{w}$ and σ. Then the posterior is proportional to the likelihood times the prior:

$$P(\boldsymbol{w}, \sigma \mid \mathrm{D}) \propto P(D \mid \boldsymbol{w}, \sigma) * P(\boldsymbol{w}, \sigma)$$

And the log of the expression is then:

$$\log P(\boldsymbol{w}, \sigma \mid \mathrm{D}) \propto \log P(D \mid \boldsymbol{w}, \sigma) + \log P(\boldsymbol{w}, \sigma)$$

The likelihood can be written as:

$$P(D \mid \boldsymbol{w}, \sigma) = \left(2\pi\sigma^2\right)^{-n/2} \exp\left(-\frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{X}\boldsymbol{w})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{w})\right)$$

Where the log-likelihood can be written as:

$$\log P(D \mid \boldsymbol{w}, \sigma) = -\frac{n}{2}\ln\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{X}\boldsymbol{w})^{\top}(\mathbf{y} - \mathbf{X}\boldsymbol{w})$$

The prior of that also resemble the penalty of a ridge regression can be written as:

$$P(\boldsymbol{w}, \sigma) = C * \exp(-\lambda \frac{(\boldsymbol{w})^T \boldsymbol{w}}{2\sigma^2})$$

Where the logarithmic form is shown as:

$$\log P(\boldsymbol{w}, \sigma) = -\lambda \frac{(\mathbf{w})^T \mathbf{w}}{2\sigma^2}$$

To obtain a optimizable $\boldsymbol{w}$ and σ one can compute by minimizing the whole expression as shown below:

$$\operatorname{argmin} \log P(\boldsymbol{w}, \sigma \mid \mathrm{D}) \propto \frac{n}{2}\ln\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{X}\boldsymbol{w})^{\top}(\mathbf{y} - \mathbf{X}\boldsymbol{w}) + \lambda \frac{(\boldsymbol{w})^T \boldsymbol{w}}{2\sigma^2}$$

2

Scale the data and divide it into training and test data (60/40). Due to this, compute all models without intercept in the following steps.

```r
# 2.2
park <- read.csv("parkinsons.csv")
park <- park %>% select(-subject.,-age,-sex,-test_time,-total_UPDRS) %>% scale() %>% data.frame()
set.seed(12345)
id <- sample(nrow(park), floor(nrow(park)*0.6 ))

train <- park %>% slice(id)
test <- park %>% slice(-id)

x_train <- train %>% select(-motor_UPDRS)
y_train <- train %>% select(motor_UPDRS)

x_test <- test %>% select(-motor_UPDRS)
y_test <- test %>% select(motor_UPDRS)
```

3

Implement 4 following functions by using basic R commands only:

a) `Loglikelihood` function that for a given parameter vector $\boldsymbol{w}$ and dispersion σ computes the log-likelihood function $\log P(D|\boldsymbol{w}, \sigma)$ the model from step 1 for the training data.

```r
# 2.3
#a)
Loglikelihood <- function(w,sigma,x,y){
  n <- nrow(x)
  x <- as.matrix(x)
  y <- as.matrix(y)
  -n/2*log(2*pi*sigma^2) - (t(y - x %*% w) %*% (y - x %*% w))/(2*sigma^2)
}
```

b) `Ridge` function that for given vector $\boldsymbol{w}$ scalar σ and scalar λ uses function from 2a and adds up a Ridge penalty to the minus log-likelihood.

```r
# 2.3
# b)

Ridge <- function(param,lambda,x,y){
  w <- matrix(param[1:ncol(x)],ncol=1)
  sigma <- param[ncol(x)+1]
  -Loglikelihood(w=w,sigma=sigma,x,y) + lambda * t(w) %*% w/(2*sigma^2)
}
```

c) `RidgeOpt` function that depends on scalar λ , uses function from 2b and function `optim()` with method="BFGS" to find the optimal $\boldsymbol{w}$ and σ for the given λ.

```r
# 2.3
# c)
RidgeOpt <- function(lambda,x,y){

result <- optim(par=c(rnorm(ncol(x)),1),
        fn = Ridge,
        lambda=lambda,
        y=y,x=x,
        method="BFGS")

list(w = result$par[1:ncol(x)], sigma = result$par[ncol(x)+1])

}
```

d) `DF` function that for a given scalar λ computes the degrees of freedom of the regression model from step 1 based on the training data.

$$DF = \mathrm{tr}\left[\mathbf{X}\left(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_{pp}\right)^{-1}\mathbf{X}^\top\right]$$

```r
# 2.3
# d)
DF <- function(lambda,x){
x<- as.matrix(x)
sum(diag(x %*% solve(t(x)%*%x + lambda*diag(ncol(x))) %*% t(x)))
}
```

4

By using function RidgeOpt, compute optimal w parameters for $\lambda = 1$, $\lambda = 100$ and $\lambda = 1000$. Use the estimated parameters to predict the motor_UPDRS values for training and test data and report the training and test MSE values.

```r
# 2.4
set.seed(12345)
l1 <- RidgeOpt(lambda = 1,x=x_train,y=y_train)
set.seed(12345)
l2 <- RidgeOpt(lambda = 100,x=x_train,y=y_train)
set.seed(12345)
l3 <- RidgeOpt(lambda = 1000,x=x_train,y=y_train)

MSE <- function(y_train,x_train,y_test,x_test,l){

y_hat_train<- as.matrix(x_train) %*%  matrix(l$w,ncol=1)
mse_train <- mean((y_train[,1] - y_hat_train)^2)

y_hat_test<- as.matrix(x_test) %*%  matrix(l$w,ncol=1)
mse_test <- mean((y_test[,1] - y_hat_test)^2)

return(data.frame(MSE_train=mse_train, MSE_test=mse_test))
}
```

Lambda	MSE_train	MSE_test
1	0.8732635	0.9291993
100	0.8768011	0.9262162
1000	0.9011675	0.9379054

Which penalty parameter is most appropriate among the selected ones?

From the table above one can conclude that the mean square of error for the test data with $\lambda = 100$ is the lowest, this indicates that this would be the most appropriate values of λ among the 1, 100 and 1000. The mean square of error for the test data seems to increase when reaching $\lambda = 1000$ which would indicate that the regularization have gone too far.

Why is MSE a more appropriate measure here than other empirical risk functions?

The goal is to minimize the true risk of the distribution, the random drawn observation from whole the data set is divided into training and test data. Mean squared error is appropriate due to the fact that one can measure the empirical risk by training the model that is using training data, hence minimizing the given loss function or with other words minimizing the empirical risk. In the same time regularization with ridge penalty is done to avoid overfitting. One can then later investigate how the computed parameters from the training data are preforming on unseen observations (test data) to see if overfitting or underfitting is occurring.

5

Use functions from step 3 to compute AIC (Akaike Information Criterion) scores for the Ridge models with values $\lambda = 1$, $\lambda = 100$ and $\lambda = 1000$ and their corresponding optimal parameters $\boldsymbol{w}$ and σ computed in step 4.

$$AIC = \log \text{likelihood}(D) + 2df(\text{model})$$

```r
# 2.5
AIC1 <- -2*Loglikelihood(w=l1$w,sigma=l1$sigma,x=park[,-1],y=park[,1]) + 2*DF(x=x_train,lambda=1)

AIC100 <- -2*Loglikelihood(w=l2$w,sigma=l2$sigma,x=park[,-1],y=park[,1]) + 2*DF(x=x_train,lambda=100)

AIC1000 <- -2*Loglikelihood(w=l3$w,sigma=l3$sigma,x=park[,-1],y=park[,1]) + 2*DF(x=x_train,lambda=1000)
```

Lambda	AIC
1	16054.61
100	16051.40
1000	16167.72

What is the optimal model according to AIC criterion?

The optimal model with $\lambda = 100$ is given by the lowest value of AIC.

What is the theoretical advantage of this kind of model selection compared to the holdout model selection done in step 4?

It's due to the fact that the AIC measures the loss of the log-likelihood of the whole data set, instead of diving the data into parts and comping the loss function afterwards. The AIC chooses the model that explains the most of the variation at the same time as minimizing the penalty.

Assignment 3 - Linear Regression and LASSO

1

Modeling fat content of meat using lasso In this assignment we are trying to use absorbance data to predict the Fat content in meat. We initially assume that Fat can be modeled as a linear regression where absorance on 100 different channels are used as features. The underlying probabilistic model can be expressed as:

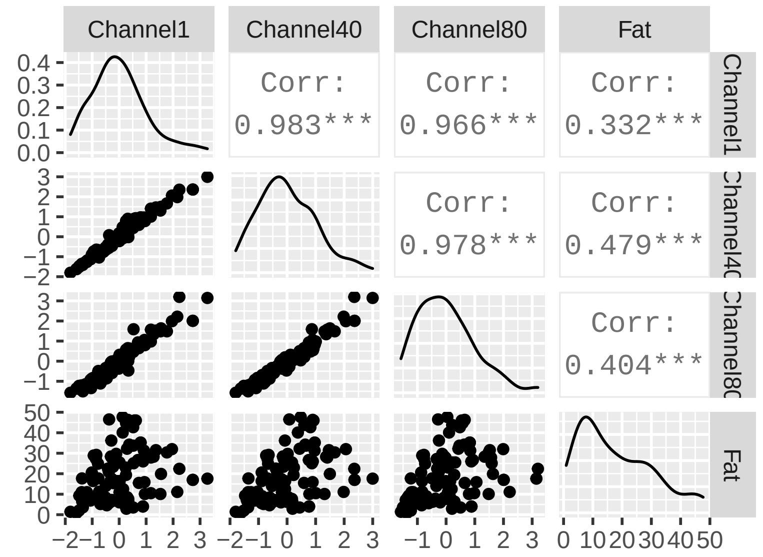
$$p(y|\boldsymbol{x}, w, \sigma^2) \sim N(w^T\boldsymbol{x}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma}} \exp -\frac{(y - w^T\boldsymbol{x})^2}{2\sigma^2}$$

which then can be estimated using a maximum likelihood estimate. Here, $\boldsymbol{x}$ are the absorbance channels 1-100 and w their respective parameters to be estimated.

To report error rates throughout the report, the root mean squared error (RMSE) is used, defined as:

$$RMSE(Y, \hat{Y}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2}$$

It is worth noting that the features are highly correlated, signalling multicollinearity might be a problem. As we now also have more features than observations, the linear regression model will probably not perform very well. Both features and target exhibit a skewness to the left.



Fitting linear regression The data has been split into test and training sets (50/50) and standarized and then a linear regression model has been fitted.

The RMSE of the training data:

```
[1] 0.0755587
```

The RMSE of the test data:

```
[1] 21.54814
```

Errors are extremely low for the training set and much higher the test set, indicating that the model is highly overfitted on the training data. The most evident cause of this is that there are more parameters than observations and that the features are highly correlated. *See appendix for code to plot training, test and target values.*

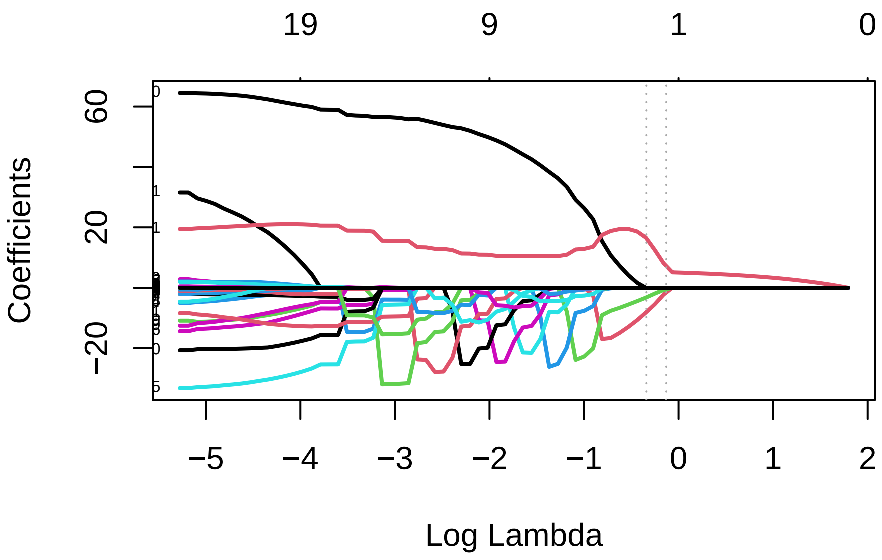
2

The objective function that should be optimized is (see Hastie et al.):

$$\hat{w}^{lasso} = argmin \left\{ \frac{1}{2} \sum_{i=1}^{N} (y_i - w_0 - \sum_{j=1}^{p} w_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |w_j| \right\}$$

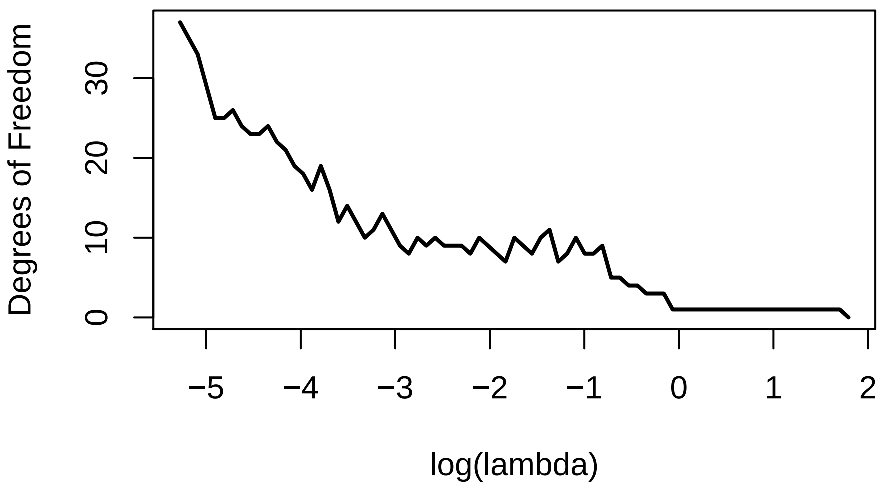
3

Fitting the lasso After fitting a lasso regression model using glmnet with an alpha-setting of 1, we can produce the following plot which shows how the coefficients vary with different levels of log lambda. The higher the penalty factor, the fewer coefficients that are non-zero. Should we want only three predictors, we can choose a log lambda corresponding to a region where there are only three non-zero coefficients (approx [-0.43,-0.13]). Interestingly, coefficients tend to go to zero and then reappear as another coefficient is eliminated. This supports our earlier finding that the multicollinearity is prominent. The variance inflation factor could be calculated to determine the severity of the problem.



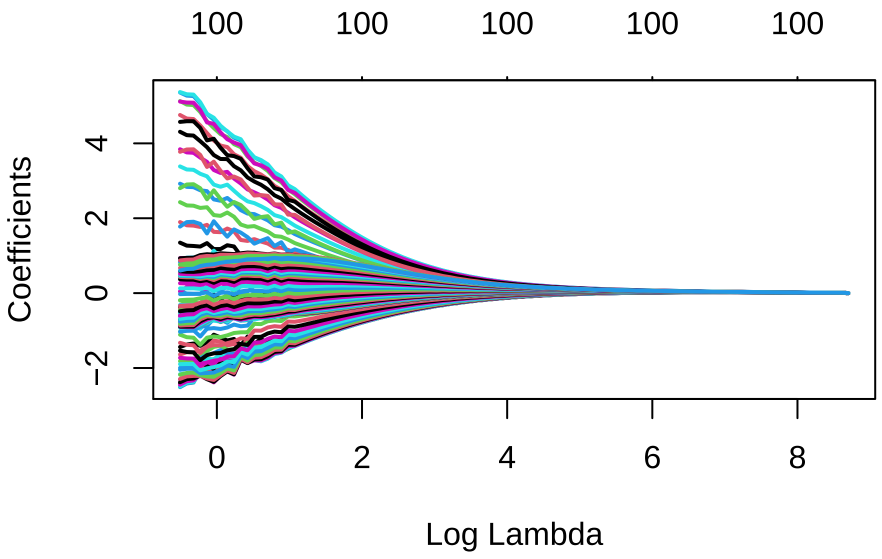
4

Degrees of Freedom The plot below shows how the degrees of freedom decreases with higher values of the regularization parameter lambda. This coincides with the number of non-zero coefficients being an exact unbiased estimate of the degrees of freedom, i.e. higher lambda means fewer non-zero components and therefore a lower degrees of freedom. The definition of the degrees of freedom should you wish to calculate it is: $df(\hat{y}) = \frac{1}{\sigma^2} \sum_{i=1}^{N} Cov(\hat{y}_i, y_i)$



5

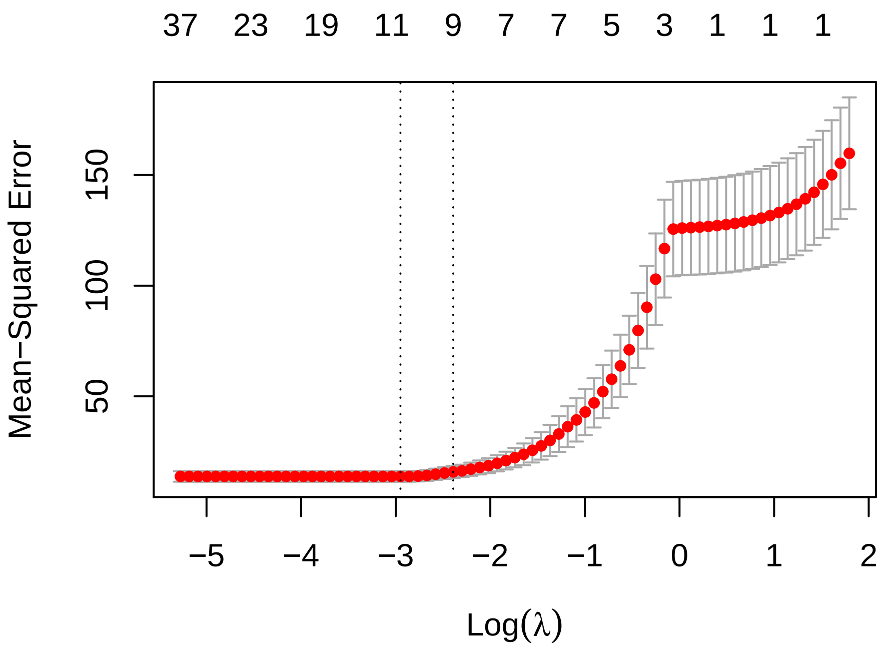
Fitting ridge model Repeating step 3 but with ridge regression gives a very different result:



It seems the ridge regression, unlike lasso, does not shrink any coefficients to zero. We are in other words left with the same number of features.

6

Cross-validation Here, cross-validation is used to find the optimal lasso model. See how the error increases when a too high log-lambda is used.



It can be determined from the plot above that the optimal value (left line) is not statistically significantly better than log lambda of -2 by looking as the error bars just so slightly overlaps. One could also extract the standard errors and intervals from the cross-validation object itself and make the calculations.

```
# 3.6 - number of non-zero coefficients in optimal model
crossval$nzero[match(crossval$lambda.min, crossval$lambda)]
```

```
s51
  9
```

```
# minimum log lambda
log(crossval$lambda.min)
```

```
[1] -2.949955
```

```
# one standard error from the minimum log lambda
log(crossval$lambda.1se)
```

```
[1] -2.391753
```

The optimal number of parameters are nine, plus the intercept. See the selected model below:

```
               Coef.
(Intercept)  17.899
Channel13   -31.768
Channel14    -3.921
Channel15    -5.542
Channel16    -0.743
Channel40    56.224
Channel41    15.547
Channel50     0.000
Channel51    -9.464
Channel52   -15.280
```

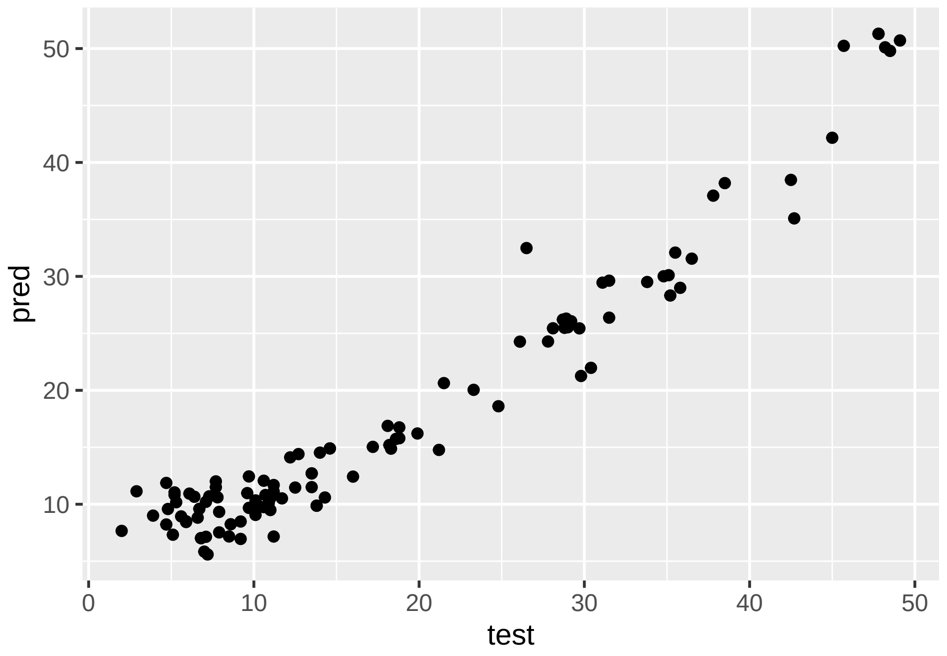
Note that the coefficient for Channel50 is very small, but non-zero.

It is possible to obtain the errors of fitted values for the training data:

```
# 3.6

# RMSE of the fitted values
sqrt(sum((y_train-as.vector(train_pred))^2) / length(y_train))
```

```
[1] 3.330301
```

Optimal model The optimal model used with the test features shows a quite strong linear relationship. Here are the fitted values of the test data plotted against the original target of the test data:



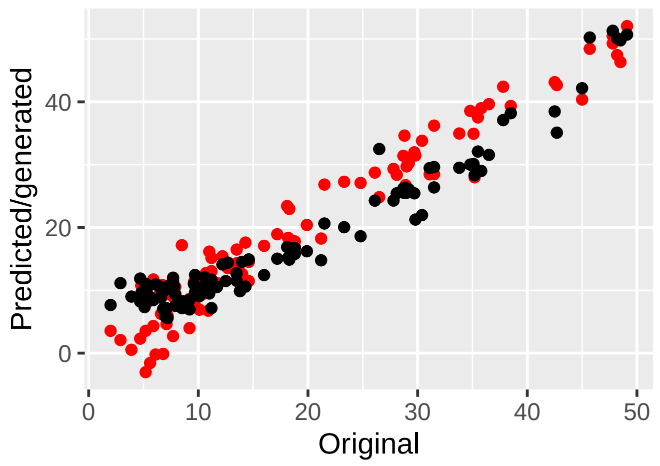
The error of the test set is given below. The difference from the error of the training data is marginal, which indicates that the model is not overfitted.

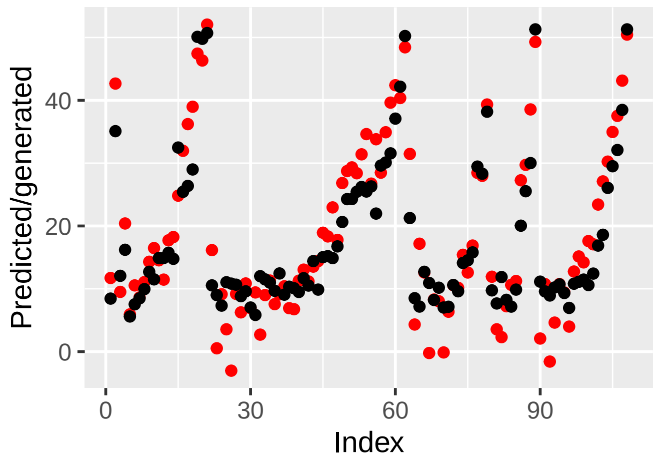
```
# 3.6

# Error of test
sqrt(sum((y_test-as.vector(lasso_test))^2) / length(y_test))
```

```
[1] 3.537068
```

7

Generating new target values Here new data is generated from the feature values of the test data and plotted against the original target values in the test data.





Newly generated points are marked in red above. Black dots are the predicted values. The match seems decent, but is not terribly exciting since we have basically used the predicted values and added some noise.

Code Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(comment = NA)

library ("kknn")

library ("dplyr")

library("reshape2")

library("ggplot2")

##  Function to Calculate the Model Accuracy

accuracy <- function(x){

  return ((sum(diag(x))/sum(rowSums(x))) * 100 )

}## End accuracy Function

##  Function to join data and probabilities, given the digit

join_data_prob <- function (df_data, fit , digit, matrix_prob){

  df <- cbind(df_data, fit, prob = matrix_prob[,colnames(matrix_prob)==digit] )

  df <- dplyr::filter(df, fit == digit)

  return(df)

}##  End join_data_prob Function
## 5.1 compute the empirical risk for the validation data as cross-entropy
## Function to calculate cross-entropy

cross_entropy <- function(par, prob){

       data <- prob[par, ]

       val <- data$actual

       log_prob <- log(data[,val+1] + 1 * exp(-15))

       return(- log_prob)

}

## Clubbing all the function into single
func_compute <- function(model, target){

  data_fit <- fitted(model)

  conf_matrix <- table (target, data_fit)
  model_accuracy <- accuracy(conf_matrix)

  model_miscalssification <- 100 - model_accuracy

  matrix_prob <- model$prob

  prob <- data.frame(matrix_prob, fit = data_fit, actual = target)

  log_prob <- lapply (1:nrow(prob),cross_entropy, prob)

  entropy <- sum(unlist(log_prob))/nrow(prob)

  return (

    list (

      fiited_val = data_fit,

      cf = conf_matrix,

      accuracy = model_accuracy,

      missclassification_error = model_miscalssification,

      prob_df = prob,

      log_prob = unlist(log_prob),

      entropy = entropy

    )

  )

}## End func_compute

## 1.1
## Reading Data
data <- read.csv("optdigits.csv", header = FALSE)

n <- dim(data)[1]

ncol <- dim(data)[2]

##  Selecting Training data (50% of total)
set.seed(12345)

id <- sample (1:n, floor(n * 0.5))

train <- data[id, ]

train_target <- train[,ncol]
```

```r
##  Selecting the data to validate (25%)

id1 <- setdiff(1:n, id)

set.seed(12345)

id2 <- sample(id1, floor(n * 0.25))

valid <- data[id2, ]

valid_target <- valid[,ncol]


##  Selecting the data to Test (25%)

id3 <- setdiff(id1, id2)

test <- data[id3, ]

test_target <- test[,ncol]

par(mfrow = c(1,1))

cnt <- c(nrow(train), nrow(valid), nrow(test))

b <- barplot(cnt,
        main = "Data Distribution",
        horiz = TRUE,
        names.arg = c("Train", "Valid", "Test"),
        cex.names=0.8,
        border = TRUE,
        space = 2,
        ylim = c(0, 10)

        )

text(x = b,  labels= as.character(c(nrow(train), nrow(valid), nrow(test))), pos = 4)
## 2
## Applying KKNN Algorithm on Training data set
cl_train <- kknn(as.factor(V65) ~ .,
                 train, test = train,
                 k = 30, kernel = "rectangular"
                 )

dt_cl_train = func_compute(cl_train, train_target)

## Applying KKNN Algorithm on TEST data set
cl_test <- kknn(as.factor(V65) ~ .,
                train, test = test,
                k = 30,
                kernel = "rectangular"
                )
```

```
dt_cl_test = func_compute(cl_test, test_target)

##  2.1
## Confusion Matrix of training data

cat("Confusion Matrix of Training Data: \n\n")

print(dt_cl_train$cf)

##  2.2
cat ("Model with k = 30")
cat("Accuracy of model with Training Data: ",
    dt_cl_train$accuracy,"%")

cat("Misclassification Error in model with Training Data: ",
    dt_cl_train$missclassification_error,"%")

cat("Accuracy of model with Test Data: ",
    dt_cl_test$accuracy,"%")

cat("Misclassification Error in model with Test Data: ",
    dt_cl_test$missclassification_error,"%")


##  3
## Classification of digit 8 in Training data

matrix_prob <- cl_train$prob

train_fit <- dt_cl_train$fiited_val

digit <-8

df_class <- join_data_prob(train, train_fit, digit, matrix_prob)

df_viz <- head(df_class[order(df_class$prob, decreasing = TRUE),],2)

df_viz <- rbind(df_viz, head(filter(df_class[order(df_class$prob,
                                                   decreasing = FALSE),],
                                                   fit == V65
                                    ),3
                             )# end head
                )# end rbind

##  Reshaping features into 8x8 matrix
temp_1 <- matrix((unlist(as.vector(df_viz[1,])[1:64])),
                 nrow = 8, ncol = 8,
                 byrow = FALSE)
temp_2 <- matrix((unlist(as.vector(df_viz[2,])[1:64])),
                 nrow = 8, ncol = 8,
                 byrow = FALSE)
temp_3 <- matrix((unlist(as.vector(df_viz[3,])[1:64])),
                 nrow = 8, ncol = 8,
```

```r
                 byrow = FALSE)
temp_4 <- matrix((unlist(as.vector(df_viz[4,])[1:64])),
                 nrow = 8, ncol = 8,
                 byrow = FALSE)
temp_5 <- matrix((unlist(as.vector(df_viz[5,])[1:64])),
                 nrow = 8, ncol = 8,
                 byrow = FALSE)

par(mfrow=c(2,3))

image(temp_1[,nrow(temp_1):1], main = "Probability = 1")
image(temp_2[,nrow(temp_2):1], main = "Probability = 1")
image(temp_3[,nrow(temp_3):1], main = "Probability = 0.33")
image(temp_4[,nrow(temp_4):1], main = "Probability = 0.33")
image(temp_5[,nrow(temp_5):1], main = "Probability = 0.36")

##  4, 5

missClassification_train = c()

missClassification_valid = c()

training_entropy <- c()

valid_entropy <- c()

for (k in 1:30){


  cl_train <- kknn(as.factor(V65) ~ .,
                   train, test = train,
                   k = k,
                   kernel = "rectangular"
                   )

  dt_cl_train = func_compute(cl_train, train_target)

  missClassification_train [k] <- dt_cl_train$missclassification_error

  training_entropy [k] <- dt_cl_train$entropy

  cl_valid <- kknn(as.factor(V65) ~ .,
                   train, test = valid,
                   k = k, kernel = "rectangular"
                   )

  dt_cl_valid = func_compute(cl_valid, valid_target)

  missClassification_valid [k] <- dt_cl_valid$missclassification_error

  valid_entropy [k] <- dt_cl_valid$entropy
```

```r
}

df_loss <- data.frame(y1 = missClassification_train,
                      y2 = missClassification_valid,
                      y3 = training_entropy,
                      y4 = valid_entropy,
                      x = 1:30
                      )

## Theme for ggplot
knn_theme <- theme(

  panel.background = element_rect(fill=NA),
  panel.border = element_rect(fill= NA),
  aspect.ratio = 0.5:0.5,

  #grid elements
  panel.grid.major = element_blank(),
  panel.grid.minor = element_blank(),
  axis.ticks = element_line(colour = 'black'),
  axis.title = element_text(face = "bold", size = 12),
  axis.line = element_line(color = 'black'),
  axis.line.x.top = element_line(color = 'black'),

  plot.title = element_text(
    size = 14,
    face = "bold",
    hjust = 0.5,
    vjust = 2),

  plot.caption = element_text(
    size = 12,
    hjust = 0.5),
)


##  4.1

df_misclassification <- melt(df_loss, id.vars = "x",

                             measure.vars= c("y1", "y2"))

 ggplot(data = df_misclassification)+

  geom_line(aes(x = x, y = value, colour = variable), size = 1.5)+

  labs( y = "MisClassification Error (%)",

        x = "K Values",

        title = "Misclassification in Training Data Vs Validation Data")+

  scale_color_manual(labels = c("missClassification_Train",
                                "missClassification_Valid"),
                     values = c("blue", "red"))+


  geom_vline(xintercept = 3, linetype = "dotted") +

  geom_vline(xintercept = 4, linetype = "dotted") +

  scale_x_continuous(breaks = 1:30) +

  knn_theme


## 4.5
## Applying KKNN Algorithm on Training data set
k = 4

k6_train <- kknn(as.factor(V65) ~ .,
                 train, test = train,
                 k = k, kernel = "rectangular"
                 )

dt_k6_train = func_compute(k6_train, train_target)

## Applying KKNN Algorithm on TEST data set
k6_valid <- kknn(as.factor(V65) ~ .,
                train, test = valid,
                k = k,
                kernel = "rectangular"
                )

dt_k6_valid = func_compute(k6_valid, valid_target)

## Applying KKNN Algorithm on TEST data set
k6_test <- kknn(as.factor(V65) ~ .,
                train, test = test,
                k = k,
                kernel = "rectangular"
                )

dt_k6_test = func_compute(k6_test, test_target)

cat("Model with K = ",k)

cat("Accuracy of  model with Training Data: ",
    dt_k6_train$accuracy,"%")

cat("Misclassification Error in model with Training Data: ",
    dt_k6_train$missclassification_error,"%")

cat("Accuracy of Model with Validation Data: ",
    dt_k6_valid$accuracy,"%")
```

```r
cat("Misclassification Error in model with  validation Data: ",
    dt_k6_valid$missclassification_error,"%")

cat("Accuracy of model with Test Data: ",
    dt_k6_test$accuracy,"%")

cat("Misclassification Error in model with Test Data: ",
    dt_k6_test$missclassification_error,"%")

df_crossEntropy <- melt(df_loss, id.vars = "x", measure.vars= c("y3", "y4"))

ggplot(data = df_crossEntropy)+

  geom_line(aes(x = x, y = value, colour = variable), size = 1.4)+

  labs( y = "mean - Cross-Entropy Error",

        x = "K Values",

        title = "Cross-Entropy in Training Data Vs Validation Data")+

  scale_color_manual(labels = c("CrossEntropy_Train", "CrossEntropy_Valid"),

                    values = c("blue", "red"))+

  geom_vline(xintercept = 6, linetype = "dotted") +

  scale_x_continuous(breaks = 1:30)+

  knn_theme
k = 6

cat("Model with K = ",k)

cat("Cross Entropy Error in model with Training Data: ",
    training_entropy[k])

cat("Cross Entropy Error in Model with Validation Data: ",
    valid_entropy[k])

# 2.2
park <- read.csv("parkinsons.csv")
park <- park %>% select(-subject.,-age,-sex,-test_time,-total_UPDRS) %>% scale() %>% data.frame()
set.seed(12345)
id <- sample(nrow(park), floor(nrow(park)*0.6 ))

train <- park %>% slice(id)
test <- park %>% slice(-id)

x_train <- train %>% select(-motor_UPDRS)
y_train <- train %>% select(motor_UPDRS)

x_test <- test %>% select(-motor_UPDRS)
```

```r
y_test <- test %>% select(motor_UPDRS)
# 2.3
#a)
Loglikelihood <- function(w,sigma,x,y){
  n <- nrow(x)
  x <- as.matrix(x)
  y <- as.matrix(y)
  -n/2*log(2*pi*sigma^2) - (t(y - x %*% w) %*% (y - x %*% w))/(2*sigma^2)
}
# 2.3
# b)

Ridge <- function(param,lambda,x,y){
  w <- matrix(param[1:ncol(x)],ncol=1)
  sigma <- param[ncol(x)+1]
  -Loglikelihood(w=w,sigma=sigma,x,y) + lambda * t(w) %*% w/(2*sigma^2)
}

# 2.3
# c)
RidgeOpt <- function(lambda,x,y){

result <- optim(par=c(rnorm(ncol(x)),1),
        fn = Ridge,
        lambda=lambda,
        y=y,x=x,
        method="BFGS")

list(w = result$par[1:ncol(x)], sigma = result$par[ncol(x)+1])

}
# 2.3
# d)
DF <- function(lambda,x){
x<- as.matrix(x)
sum(diag(x %*% solve(t(x)%*%x + lambda*diag(ncol(x))) %*% t(x)))
}
# 2.4
set.seed(12345)
l1 <- RidgeOpt(lambda = 1,x=x_train,y=y_train)
set.seed(12345)
l2 <- RidgeOpt(lambda = 100,x=x_train,y=y_train)
set.seed(12345)
l3 <- RidgeOpt(lambda = 1000,x=x_train,y=y_train)

MSE <- function(y_train,x_train,y_test,x_test,l){

y_hat_train<- as.matrix(x_train) %*%  matrix(l$w,ncol=1)
mse_train <- mean((y_train[,1] - y_hat_train)^2)

y_hat_test<- as.matrix(x_test) %*%  matrix(l$w,ncol=1)
mse_test <- mean((y_test[,1] - y_hat_test)^2)
```

```
return(data.frame(MSE_train=mse_train, MSE_test=mse_test))
}

knitr::kable(
data.frame(Lambda=c(1,100,1000),rbind(
MSE(y_train=y_train,x_train = x_train,
    y_test = y_test, x_test =  x_test, l=l1),
MSE(y_train=y_train,x_train = x_train,
    y_test = y_test, x_test =  x_test, l=l2),
MSE(y_train=y_train,x_train = x_train,
    y_test = y_test, x_test =  x_test, l=l3))))
# 2.5
AIC1 <- -2*Loglikelihood(w=l1$w,sigma=l1$sigma,x=park[,-1],y=park[,1]) + 2*DF(x=x_train,lambda=1)

AIC100 <- -2*Loglikelihood(w=l2$w,sigma=l2$sigma,x=park[,-1],y=park[,1]) + 2*DF(x=x_train,lambda=100)

AIC1000 <- -2*Loglikelihood(w=l3$w,sigma=l3$sigma,x=park[,-1],y=park[,1]) + 2*DF(x=x_train,lambda=1000)




knitr::kable(data.frame(Lambda=c(1,100,1000),AIC=c(AIC1,AIC100,AIC1000)))
knitr::opts_chunk$set(echo = TRUE)
#setwd(dir = "C:/Users/Henrik Olofsson/Dropbox/732A99 Machine Learning/Lab 1-1/")

library(dplyr)
library(ggplot2)

# Reading data
tecator <- read.csv(file = "tecator.csv", row.names = "Sample")

# Splitting into test and train (50/50)
n <- dim(tecator)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- tecator[id, ]
test <- tecator[-id, ]

train <- dplyr::select(train, -c("Protein", "Moisture"))
test <- dplyr::select(test, -c("Protein", "Moisture"))

train <- as.data.frame(cbind(scale(train[,1:ncol(train)-1]), "Fat" = train[,ncol(train)]))
test <- as.data.frame(cbind(scale(test[,1:ncol(test)-1]), "Fat" = test[,ncol(test)]))


# 3
GGally::ggpairs(train[, c(1, 40, 80, 101)])
# 3.1
lm_mod <- lm(formula = Fat~., data = train)
lm_pred <- predict(lm_mod, newdata = test)
# 3.1

# Error (RMSE) of training data
```

```r
sqrt(sum((train$Fat-as.vector(lm_mod$fitted.values))^2) / nrow(train))
# 3.1

# Error (RMSE) of test set
sqrt(sum((test$Fat-as.vector(lm_pred))^2) / length(lm_pred))

# 3.1

# Plot of training, test and original target values.
# Target values have been shifted slightly to make it possible to distinguish them
# from the predicted values from the training data.
p <- ggplot(data = as.data.frame(lm_pred)) +
  geom_point(aes(1:length(lm_pred), lm_pred)) +
  geom_point(
    data = as.data.frame(lm_mod$fitted.values),
    aes(1:length(lm_mod$fitted.values), lm_mod$fitted.values),
    colour = "purple"
  ) +
  geom_point(data = as.data.frame(train$Fat),
             aes(1:nrow(train), train$Fat + 2.5),
             colour = "pink")
p

# 3.3

# splitting features and target
x_train <- as.matrix(dplyr::select(train, -c("Fat")))
y_train <- train$"Fat"

# fitting lasso
lasso <- glmnet::glmnet(x = x_train, y = y_train, alpha = 1)

# plotting coefficients
plot(lasso, xvar = "lambda", lwd = 2, label = TRUE)
abline(v = -.34, col = "darkgray", lty = 3)
abline(v = -.13, col = "darkgray", lty = 3)
# 3.4
loglambda <- log(lasso$lambda)
plot(loglambda, lasso$df, type = "l", lwd = 2, ylab = "Degrees of Freedom", xlab = "log(lambda)")

# 3.5
ridge <- glmnet::glmnet(x = x_train, y = y_train, alpha = 0)
plot(ridge, xvar = "lambda", lwd = 2)

# 3.6

# cross-validation
set.seed(12345)
crossval <- glmnet::cv.glmnet(x = x_train, y = y_train)

plot(crossval)

# 3.6 - number of non-zero coefficients in optimal model
```

```r
crossval$nzero[match(crossval$lambda.min, crossval$lambda)]

# minimum log lambda
log(crossval$lambda.min)

# one standard error from the minimum log lambda
log(crossval$lambda.1se)
# 3.6 - optimal model
temp <- coef(crossval, s="lambda.min")
coefs <- as.matrix(round(temp[temp != 0],3))
rownames(coefs) <- rownames(temp)[which(as.matrix(temp) != 0)]
colnames(coefs) <- "Coef."
coefs

# 3.6

# using optimal model to fit data
m1 <- glmnet::glmnet(x = x_train, y = y_train, lambda = crossval$lambda.min, alpha = 1)
train_pred<- glmnet::predict.glmnet(m1, newx = x_train, s = "lambda.min")
# 3.6

# RMSE of the fitted values
sqrt(sum((y_train-as.vector(train_pred))^2) / length(y_train))

# 3.6

# splitting
x_test <- as.matrix(dplyr::select(test, -c("Fat")))
y_test <- test$Fat

lasso_test <- glmnet::predict.glmnet(m1, newx = x_test)

df <- data.frame("test" = y_test, "pred" = as.vector(lasso_test))

ggplot(data = df, aes(x = test, y = pred)) +
  geom_point()

# 3.6

# Error of test
sqrt(sum((y_test-as.vector(lasso_test))^2) / length(y_test))

# 3.7

# standard deviation of the residuals from the training data
sigma <- sqrt(var(y_train - as.vector(train_pred)))

# generating new target values using the mean of the feature values
# and the standard deviation from the training data
new_target <- rnorm(length(y_test), y_test, sigma)

# plotting
df2 <- data.frame(y_test, new_target)
```

```r
names(df2) <- c("Fat (test data)", "Fat (generated)")

# generated data in red
ggplot(data = df2, aes(x = y_test, y = new_target)) +
  geom_point(colour = "red") +
  geom_point(data = df, aes(x = test, y = pred)) +
  labs(x = "Original", y = "Predicted/generated")

ggplot(data = df2, aes(x = 1:length(new_target), y = new_target)) +
  geom_point(colour = "red") +
  geom_point(data = df, aes(x = 1:length(pred), y = pred)) +
  labs(x = "Index", y = "Predicted/generated")
```