1 Lab 1 Task 1: K-Nearest Neighbor Classification

The data file optdigits.csv contains information about normalized bitmaps of handwritten digits from a preprinted form from a total of 43 people. The data were first derived as 32x32 bitmaps which were then divided into nonoverlapping blocks of 4x4, and the number of on pixels was counted in each block. This generated a resulting image of size 8x8 where each element is an integer in the range 0..16. Accordingly, each row in the data file is a sequence corresponding to an 8x8 matrix, and the last element shows the actual digit from 0 to 9.

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. Use the training data to fit a 30-nearest neighbor classifier with the function kknn() and kernel="rectangular" from the kknn package, 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.
2. 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 the highest and lowest probabilities of the correct class). Reshape features for each of these cases as an 8x8 matrix and visualize the corresponding digits (e.g., using the heatmap() function with parameters Colv=NA and Rowv=NA). Comment on whether these cases seem to be hard or easy to recognize visually.
3. Fit K-nearest neighbor classifiers to the training data for different values of $K = 1, 2, \dots, 30$, 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. Finally, estimate the test error for the model having the optimal K, compare it with the training and validation errors, and make necessary conclusions about the model quality.
4. Fit K-nearest neighbor classifiers to the training data for different values of $K = 1, 2, \dots, 30$, compute the error for the validation data as cross-entropy (when computing the log of probabilities, add a small constant within the log, e.g., $1e{-}15$, to avoid numerical problems), and plot the dependence of the validation error on the value of K. What is the optimal K value here? Assuming that the response has a multinomial distribution, why might the cross-entropy be a more suitable choice of error function than the misclassification error for this problem?

```r
############################ PART 1 ############################

library(kknn)

data_frame = read.csv("../optdigits.csv", header = FALSE)

# Rename the last column to "label", represents the actual digit
colnames(data_frame)[65] = "label"

# Convert the label column to a factor for classification
data_frame$label = as.factor(data_frame$label) # Treat as values categories (0 - 9)
```

```r
n = dim(data_frame)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.5))
data_train = data_frame[id, ]

id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n * 0.25))
data_validation = data_frame[id2, ]

id3 = setdiff(id1, id2)
data_test = data_frame[id3, ]

############################ PART 2 ############################

# Performs k-nearest neighbor classification of a test set using a training set.
model_test = kknn(formula = label ~ ., train = data_train, test = data_test, k = 30, kernel =
↪   "rectangular")
model_train = kknn(formula = label ~ ., train = data_train, test = data_train, k = 30, kernel =
↪   "rectangular")

# Generate predictions for test and training data
predictions_test = predict(model_test)
predictions_train = predict(model_train)

# Generate confusion matrix (classification) comparing predictions with true data.
conf_matrix_test = table(predictions_test, data_test$label)
conf_matrix_train = table(predictions_train, data_train$label)
conf_matrix_test
conf_matrix_train

# Function to calculate the misclassification rate
# X = true labels, X1 = predictions
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
missclass(predictions_test, data_test$label)
missclass(predictions_train, data_train$label)

# predictions_test   0   1   2   3   4   5   6   7   8   9
# 0  77   0   0   0   0   0   0   0   0   0
# 1   0  81   0   0   0   1   0   0   7   1
# 2   0   2  98   0   0   1   0   0   0   1
# 3   0   0   0 107   0   0   0   1   1   1
# 4   1   0   0   0  94   0   0   0   0   0
# 5   0   0   0   2   0  93   0   0   0   0
# 6   0   0   0   0   2   2  90   0   0   0
# 7   0   0   0   0   6   1   0 111   0   1
# 8   0   0   3   1   2   0   0   0  70   0
# 9   0   3   0   1   5   5   0   0   0  85
# predictions_train   0   1   2   3   4   5   6   7   8   9
# 0 202   0   0   0   1   0   0   0   0   1
# 1   0 179   1   0   3   0   2   3  10   3
# 2   0  11 190   0   0   0   0   0   0   0
# 3   0   0   0 185   0   1   0   0   2   5
# 4   0   0   0   0 159   0   0   0   0   2
# 5   0   0   0   1   0 171   0   0   0   0
# 6   0   0   0   0   0   0 190   0   2   0
# 7   0   1   1   1   7   1   0 178   0   3
# 8   0   1   0   0   1   0   0   1 188   3
# 9   0   3   0   1   4   8   0   0   2 183


# > missclass(predictions_test, data_test$label)
# [1] 0.05329154
# > missclass(predictions_train, data_train$label)
# [1] 0.04500262

# The overall quality of the models predictions is quite good, which can be seen
# above. For the test and training data set the misclassification rates are only
# 0.05329154 and 0.04500262 respectively. When checking the confusion matrix for
# the training dataset there are some values that stand out. These are the
# numbers 1, 4, 5, 8 and 9. These were wrongly classified the most by the model.
# This is probably because some handwritten numbers look the same when using a
# low resolution image.

########################### PART 3 ###########################

# Get the probability matrix from the trained model
prob_matrix = model_train$prob

# Identify rows with the label "8" in the training dataset
label_8_indices = which(data_train$label == "8")

# Extract probabilities of the true label "8" for these rows
label_8_prob = prob_matrix[label_8_indices, "8"]

# Sort the indices by descending probabilities
sorted_indices = order(label_8_prob, decreasing = TRUE)

# Select the indices of the two easiest and three hardest classifications
easiest_to_classify = label_8_indices[sorted_indices[1:2]]
hardest_to_classify =
↪   label_8_indices[sorted_indices[(length(sorted_indices)-2):length(sorted_indices)]]

# Reshape to 8x8 matrices to visualize the digit with heat map
for (i in 1:2) {
  easiest_case = matrix(as.numeric(data_train[easiest_to_classify[i], 1:64]), nrow = 8, ncol =
  ↪   8, byrow=TRUE)
  heatmap(easiest_case, Rowv = NA, Colv = NA, main = paste("Easiest Case", i))
```

```r
}
for (i in 1:3) {
  hardest_case = matrix(as.numeric(data_train[hardest_to_classify[i], 1:64]), nrow = 8, ncol =
  ↪   8, byrow=TRUE)
  heatmap(hardest_case, Rowv = NA, Colv = NA, main = paste("Hardest Case", i))
}

# The easiest two classifications are simple to recognize while the three hardest are almost
↪   impossible.

########################### PART 4 ###########################

# Decide how many data points from neighboring data you want to consider when making
↪   predictions.
min_k = 1
max_k = 30
k_values = min_k:max_k
missclass_vector = c()
missclass_vector_validation = c()

for (k_value in k_values) {
  model_train = kknn(formula = label ~ ., train = data_train, test = data_train, k = k_value,
  ↪   kernel = "rectangular")
  model_validation = kknn(formula = label ~ ., train = data_train, test = data_validation, k =
  ↪   k_value, kernel = "rectangular")

  predict_train = predict(model_train)
  predict_validation = predict(model_validation)

  prob_validation = model_validation$prob

  missclass_value = missclass(predict_train, data_train$label)
  missclass_value_validation = missclass(predict_validation, data_validation$label)

  missclass_vector = append(missclass_vector, missclass_value)
  missclass_vector_validation = append(missclass_vector_validation,
  ↪   missclass_value_validation)
}

plot(k_values, missclass_vector, col = "red", xlab = "k (Number of neighbors)", ylab =
↪   "Misclassification rate", main = "misclassification rate vs k value", ylim = c(0, 0.05))
points(k_values, missclass_vector_validation, col = "blue")

missclass_vector_validation[which.min(missclass_vector_validation)]

legend("bottomright", legend = c("Training", "Validation"), col = c("red", "blue"), lty = 1)

# When the K values are small the complexity of the model is high and there is a big risk for
↪   overfitting. While when the K values are big the complexity of the model decreases and
↪   there is risk for underfitting. By looking at the figure we can see that for low K values the
↪   red and blue model is better while for big K values both models is almost equally as bad.
#
```

```
# The most optimal K value for the blue model is K = 3 then the misclassification value is
↪   0.02513089 and for the red model is K = 1 and misclassification value is 0.00. This is
↪   because the red model is trained and tested on the same dataset. This makes the model
↪   look very good for low K values but in reality its overfitted. The blue model has worse
↪   misclassification value but its not trained and tested on the same dataset so it shows a
↪   more realistic rate for different K values.

########################### PART 5 ###########################

LOG_CONSTANT = 1e-15
cross_entropy = c()

for (k_value in k_values) {
  model_train = kknn(formula = label ~ ., train = data_train, test = data_train, k = k_value,
  ↪   kernel = "rectangular")
  model_validation = kknn(formula = label ~ ., train = data_train, test = data_validation, k =
  ↪   k_value, kernel = "rectangular")

  prob_validation = model_validation$prob

  # Cross entropy
  cross_entropy_k = 0
  for (i in 1:nrow(data_validation)) {
    true_label = data_validation$label[i] # Gets label for each row
    true_prob = prob_validation[i, as.numeric(true_label)] # Get prob for current row and true
    ↪   label
    cross_entropy_k = cross_entropy_k - log(true_prob + LOG_CONSTANT) # calc cross
    ↪   entropy
  }
  cross_entropy = c(cross_entropy, cross_entropy_k / nrow(data_validation))
}

plot(k_values, cross_entropy, xlab = "k (Number of neighbors)", ylab = "Cross-entropy loss",
↪   main = "Cross-Entropy Loss vs. k", ylim = c(0, 1.0))
which.min(cross_entropy)

# From the figure we can see that the optimal K value is 6, because it has the lowest loss.
↪   The cross entropy is a more suitable choice for the error function because it evaluates
↪   the quality of the predicted probabilities. Compared to the misclassification which only
↪   cares if the predicted digit was correct or not, and not how high probability that the true
↪   digit actually has.
```

2 Lab 1 Task 2: Linear Regression and Ridge Regression

The data file parkinson.csv is composed of a range of biomedical voice measurements from 42 people with early-stage Parkinson's disease recruited to a six-month trial of a telemonitoring device for remote symptom progression monitoring. The purpose is to predict Parkinson's disease symptom score (motor UPDRS) from the following voice characteristics:

- **Jitter**: (%), Jitter(Abs), Jitter:RAP, Jitter:PPQ5, Jitter:DDP - Several measures of variation in fundamental frequency.
- **Shimmer**: Shimmer(dB), Shimmer:APQ3, Shimmer:APQ5, Shimmer:APQ11, Shimmer:DDA - Several measures of variation in amplitude.
- **NHR**, **HNR** - Two measures of the ratio of noise to tonal components in the voice.
- **RPDE** - A nonlinear dynamical complexity measure.
- **DFA** - Signal fractal scaling exponent.
- **PPE** - A nonlinear measure of fundamental frequency variation.

1. Divide the data into training and test sets (60/40) and scale it appropriately. In the following steps, assume that motor_UPDRS is normally distributed and is a function of the voice characteristics. Since the data are scaled, no intercept is needed in the modeling.
2. Compute a linear regression model from the training data, estimate training and test MSE, and comment on which variables contribute significantly to the model.
3. Implement the following functions using basic R commands only (no external packages):
   a) **LogLikelihood**: A function for a given parameter vector θ and dispersion σ computes the log-likelihood function log $P(T|\theta, \sigma)$ for the stated model and the training data.
   b) **Ridge**: A function that, for a given vector θ, scalar σ, and scalar λ, adds a Ridge penalty $\lambda||\theta||^2$ to the minus log-likelihood.
   c) **RidgeOpt**: A function that dependso on the scalar λ, uses function 3b and function optim() with method="BFGS" to find the optimal θ and σ for a given λ.
   d) **DF (Degrees of Freedom)**: function that for given scalar λ computes the degrees of freedom of the ridge model based on the training data.
4. Using the Ridge optimization function, compute the optimal θ parameters for $\lambda = 1$, $\lambda = 100$, and $\lambda = 1000$. Use the estimated parameters to predict motor_UPDRS values for training and test data and report the training and test MSE values. Determine which penalty parameter is most appropriate among the selected ones. Compute and compare the degrees of freedom of these models and make appropriate conclusions.

```
# _______ TASK 1 _______

data <- read.csv('parkinsons.csv')
labels <- colnames(data)

parameters<-c("Jitter...", "Jitter.Abs.", "Jitter.RAP", "Jitter.PPQ5", "Jitter.DDP",
          "Shimmer", "Shimmer.dB.", "Shimmer.APQ3", "Shimmer.APQ5", "Shimmer.APQ11",
          "Shimmer.DDA", "NHR", "HNR", "RPDE", "DFA", "PPE")
parameters_y<-c(parameters, "motor_UPDRS")

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train=data[id,]

valid=data[-id,]

library(caret)
```

```r
scaler=preProcess(train)
trainS=predict(scaler,train)
validS=predict(scaler,valid)

trainS = trainS[, colnames(trainS) %in% parameters_y]
validS = validS[, colnames(validS) %in% parameters_y]

trainS_without_y_matrix = as.matrix(subset(trainS, select=-motor_UPDRS))
validS_without_y_matrix = as.matrix(subset(validS, select=-motor_UPDRS))

# ____________ TASK 2 _____________

y = trainS$motor_UPDRS
y_validS = validS$motor_UPDRS

fit=lm(y ~ . - motor_UPDRS, data=trainS)

coeff = coefficients(fit)[-1] # model coefficients, intercept not needed since the data is scaled
confint(fit, level=0.95) # CIs for model parameters
fitted(fit) # predicted values
residuals_trainS = residuals(fit) # residuals

# _________________________

predicted_validS = predict(fit, newdata=validS)
residuals_validS <- validS$motor_UPDRS - predicted_validS

# ANSWER:
mean(residuals_trainS^2) # MSE for trainS = 0.8785431
mean(residuals_validS^2) # MSE for validS = 0.9354477

summary(fit)

# ANSWER:
# summary(fit) will print some stuff. The three stars next to some variables mean that there is
↪   a high correlation
# between them and the target variable motor_UPDRS.
# Answer: Jitter.Abs., Shimmer.APQ5, Shimmer.APQ11, NHR, HNR, DFA and PPE

# ________ TASK 3 ________
Loglikelihood <- function(theta, sigma) # linear regression gives normal distribution
{
  - nrow(trainS) / 2 * log(2 * pi) - nrow(trainS) / 2 * log(sigma^2) - 1/(2*sigma^2) * sum((y -
  ↪   trainS_without_y_matrix %*% theta)^2)
}

Ridge <- function(theta, lambda, sigma)
{
  -Loglikelihood(theta, sigma) + lambda * sum(theta^2)
}

RidgeOpt <- function(lambda)
```

```
{
  optim(coeff, fn = Ridge, lambda = lambda, sigma = 1, method = "BFGS")
}

DF <- function(lambda)
{
  X = trainS_without_y_matrix
  sum(diag(X %*% solve(t(X) %*% X + lambda * diag(ncol(X))) %*% t(X)))
}
# _____ TASK 4 ______________

lambda_1 = RidgeOpt(1)$par
lambda_100 = RidgeOpt(100)$par
lambda_1000 = RidgeOpt(1000)$par

# _____________

# Use the estimated parameters to predict the motor_UPDRS values for training data
predict_trainS_motor_UPDRS_lambda_1 = trainS_without_y_matrix %*% lambda_1
predict_trainS_motor_UPDRS_lambda_100 = trainS_without_y_matrix %*% lambda_100
predict_trainS_motor_UPDRS_lambda_1000 = trainS_without_y_matrix %*% lambda_1000

# calculate the MSE
MSE_trainS_lambda_1 = mean((predict_trainS_motor_UPDRS_lambda_1 - y)^2)
MSE_trainS_lambda_100 = mean((predict_trainS_motor_UPDRS_lambda_100 - y)^2)
MSE_trainS_lambda_1000 = mean((predict_trainS_motor_UPDRS_lambda_1000 - y)^2)

# _____________

# Use the estimated parameters to predict the motor_UPDRS values for test data
predict_validS_motor_UPDRS_lambda_1 = validS_without_y_matrix %*% lambda_1
predict_validS_motor_UPDRS_lambda_100 = validS_without_y_matrix %*% lambda_100
predict_validS_motor_UPDRS_lambda_1000 = validS_without_y_matrix %*% lambda_1000

# Calculate the MSE
MSE_validS_lambda_1 = mean((predict_validS_motor_UPDRS_lambda_1 - y_validS)^2)
MSE_validS_lambda_100 = mean((predict_validS_motor_UPDRS_lambda_100 -
↪   y_validS)^2)
MSE_validS_lambda_1000 = mean((predict_validS_motor_UPDRS_lambda_1000 -
↪   y_validS)^2)

DF_lambda_1 = DF(1)
DF_lambda_100 = DF(100)
DF_lambda_1000 = DF(1000)

# _____________

# ANSWER:
MSE_trainS_lambda_1 # 0.8786332
MSE_trainS_lambda_100 # 0.8850982
MSE_trainS_lambda_1000 # 0.9232605
```

```
MSE_validS_lambda_1 # 0.9349501
MSE_validS_lambda_100 # 0.9324709
MSE_validS_lambda_1000 # 0.9554836

DF_lambda_1 # 13.86074
DF_lambda_100 # 9.924887
DF_lambda_1000 # 5.643925

# We see that MSE_validS_lambda_100 is the lowest which
# means that the lambda = 100 is the most suitable choice. In this case it can be
# seen that a higher degree of freedom is not always the best choice since it
# can lead to overfitting.
```

3 Lab 1 Task 3: Logistic Regression and Basis Function Expansion

The data file pima-indians-diabetes.csv contains information about the onset of diabetes within 5 years in Pima Indians given medical details. The variables are (in the same order as in the dataset):

1. Number of times pregnant.
2. Plasma glucose concentration 2 hours into an oral glucose tolerance test.
3. Diastolic blood pressure (mm Hg).
4. Triceps skinfold thickness (mm).
5. 2-Hour serum insulin (μ U/ml).
6. Body mass index (weight in kg/(height in m)^2).
7. Diabetes pedigree function.
8. Age (years).
9. Diabetes (0 = no or 1 = yes).

1. Make a scatterplot showing Plasma glucose concentration on Age where observations are colored by Diabetes levels. Do you think that Diabetes is easy to classify by a standard logistic regression model that uses these two variables as features? Motivate your answer.
2. Train a logistic regression model with $y =$ Diabetes as target, $x_1 =$ Plasma glucose concentration, and $x_2 =$ Age as features, and make a prediction for all observations by using $r = 0.5$ as the classification threshold.
   - Report the probabilistic equation of the estimated model (i.e., how the target depends on the features and the estimated model parameters probabilistically).
   - Compute the training misclassification error and make a scatterplot of the same kind as in step 1 but showing the predicted values of Diabetes as a color instead.
   - Comment on the quality of the classification by using these results.
3. Use the model estimated in step 2 to:
   a) Report the equation of the decision boundary between the two classes.
   b) Add a curve showing this boundary to the scatterplot in step 2.

Comment whether the decision boundary seems to capture the data distribution well.

4. Make the same kind of plots as in step 2 but use thresholds $r = 0.2$ and $r = 0.8$. By using these plots, comment on what happens with the prediction when the r value changes.
5. Perform a basis function expansion trick by computing new features:

$$z_1 = x_1^4, \quad z_2 = x_1^3 x_2, \quad z_3 = x_1^2 x_2^2, \quad z_4 = x_1 x_2^3, \quad z_5 = x_2^4$$

Add them to the dataset and compute a logistic regression model with y as target and $x_1, x_2, z_1, \dots, z_5$ as features.

- Create a scatterplot of the same kind as in step 2 for this model.
- Compute the training misclassification rate.
- What can you say about the quality of this model compared to the previous logistic regression model?
- How has the basis expansion trick affected the shape of the decision boundary and the prediction accuracy?

```r
library(ggplot2)

# Set seed to 12345
set.seed(12345)

# Import data
data <- read.csv("../pima-indians-diabetes.csv", header = FALSE)

# Add headers to data
colnames(data) <- c("times_pregnant",
                    "plasma",
                    "blood_pressure",
                    "skinfold_thickness",
                    "insulin",
                    "bmi",
                    "diabetes_pedigree",
                    "age",
                    "diabetes")

# ___________ TASK 1 _________________

# Plot data where age is the x axis, plasma is the y axis and color is based
# on diabetes value
ggplot(data, aes(x = age, y = plasma, color = diabetes)) +
  geom_point() + scale_color_gradient(low = "green", high = "red") +
  labs(title = "Task 1",
       x = "Age",
       y = "Plasma Glucose Concentration",
       color = "Diabetes") +
  theme_minimal()

# By observing the result of the task, we deem that it is difficult to classify diabetes by a
↪ standard logistic regression model that uses these two variables as features. When
↪ looking at the graph it becomes quite clear that there isn't a strong enough connection
↪ between plasma levels and age to determine if a person has diabetes or not. There are
↪ for example several people between age 60 and 70 with a plasma level above 150
↪ whom do not have diabetes. Meanwhile there are also young people below the age of
↪ 30 with a plasma level below 100 whom have diabetes. In conclusion, we determine that
↪ diabetes is hard to classify using these two variable. However, we have observed that
↪ diabetes seems more common for older ages and higher plasma levels.
```

```r
# __________ TASK 2 ____________________

model = glm(diabetes~age + plasma,data=data, family = binomial)
predictions <- predict(model, type = "response")

#Initialize all different R values to be used for future plotting.
r1 <- 0.5


#Calculate binary predictions based on all three r values.
binary_predictions_1 <- ifelse(predictions >= r1, 1, 0)

#Function to calculate missclassification error.
missclass = function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}

#Calculate missclassification error.
misclassification_error = missclass(binary_predictions_1, data$diabetes) # 0.2630208

# ANSWER:
# the family variable is set to binomial because the response variable follows a binomial
↪   distribution, since it is a binary variable.

#Plot binary predictions and the decision boundary where r = 0.5.
ggplot(data, aes(x = age, y = plasma, color = binary_predictions_1)) +
  geom_point() + scale_color_gradient(low = "green", high = "red") +
  labs(title = "Diabetes Predictions r = 0.5",
       x = "Age",
       y = "Plasma Glucose Concentration",
       color = "Diabetes") +
  theme_minimal()


# ANSWER:
# Probabilistic model and figure is under the code.

# _________ TASK 3 _________________

#Obtain coefficients from model.
b0 <- coef(model)[1]
b1 <- coef(model)[2]
b2 <- coef(model)[3]

#Arrange X1 in order from smallest to largest.
x1 <- seq(min(data$age), max(data$age))

# Calculate x2 values for the decision boundary.
# b_0 + b_1 * x_1 + b_2 * x_2 = 0 => x2
x2 <- -(b0 + b1 * x1) / b2
```

```
#Create the decision boundary line based on x1 and x2 values
decision_boundary <- data.frame(x1 = x1, x2 = x2)

#Plot binary predictions and the decision boundary where r = 0.5.
ggplot(data, aes(x = age, y = plasma, color = binary_predictions_1)) +
  geom_point() + scale_color_gradient(low = "green", high = "red") +
  labs(title = "Diabetes Predictions r = 0.5 + decision boundary",
       x = "Age",
       y = "Plasma Glucose Concentration",
       color = "Diabetes") +
  geom_line(data = decision_boundary, aes(x = x1, y = x2), color = "blue", linewidth = 1) +  #
  ↪    Add decision boundary
  theme_minimal()

# ANSWER:
# It seems to catch the data distribution well since people predicted to suffer from diabetes or
↪    above it or on it and people predicted not to suffer from diabetes are on it or below it.
↪    The decision boundary can be observed in figure.

# _______ TASK 4 _______________

r2 <- 0.2
r3 <- 0.8

binary_predictions_2 <- ifelse(predictions >= r2, 1, 0)
binary_predictions_3<- ifelse(predictions >= r3, 1, 0)

ggplot(data, aes(x = age, y = plasma, color = binary_predictions_2)) +
  geom_point() + scale_color_gradient(low = "green", high = "red") +
  labs(title = "Diabetes Predictions r = 0.2",
       x = "Age",
       y = "Plasma Glucose Concentration",
       color = "Diabetes") +
  theme_minimal()

ggplot(data, aes(x = age, y = plasma, color = binary_predictions_3)) +
  geom_point() + scale_color_gradient(low = "green", high = "red") +
  labs(title = "Diabetes Predictions, r = 0.8",
       x = "Age",
       y = "Plasma Glucose Concentration",
       color = "Diabetes") +
  theme_minimal()

# ANSWER:
# By observing these plots it becomes evident that r decides how certain our model has to
↪    bee in order to decide on whether a person suffers from diabetes or not. In other words,
↪    a person is predicted to suffer from diabetes if the predicted probability of the model for
↪    that person is greater than r. This results in a greater amount of people predicted to
↪    suffer from diabetes by the model for lower r values.

# _____________ TASK 5 ________________
```

```
data$z1 = data$plasma^4
data$z2 = data$plasma^3 * data$age
data$z3 = data$plasma^2 * data$age^2
data$z4 = data$plasma * data$age^3
data$z5 = data$age^4

model2 <- glm(diabetes ~ plasma + age + z1 + z2 + z3 + z4 + z5, family = binomial, data =
↪   data)

predictions <- predict(model2, newdata = data, type = "response")
r <- 0.5
binary_predictions <- ifelse(predictions >= r, 1, 0)

ggplot(data, aes(x = age, y = plasma, color = binary_predictions)) +
  geom_point() + scale_color_gradient(low = "green", high = "red") +
  labs(title = "Diabetes Predictions r = 0.5",
       x = "Age",
       y = "Plasma Glucose Concentration",
       color = "Diabetes") +
  theme_minimal()

misclassification_error <- missclass(data$diabetes, binary_predictions)
print(misclassification_error) # 0.2447917

# ANSWER:
# This is a small improvement compared to the missclassification error observed in task 2,
↪   but one could argue that the two answers are in a similar range and neither one is
↪   particularly good. even though this new model has a slightly smaller missclassification
↪   error than the previous one, it has one large issue which is that the prediction rate for
↪   diabetes decreases for higher ages. This seems like an unwanted behaviour since
↪   diabetes is usually more common in higher ages, but this prediction model basically
↪   predicts that people above a certain age (around 65) never have diabetes.
```

Probabilistic equation of the model in task 2:

$$\text{P(diabetes| age, plasma)} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 * age + \beta_2 * plasma)}}$$

4 Lab 2 Task 1: Tecator Dataset Analysis

The file tecator.csv contains the results of a study aimed at investigating whether a near-infrared absorbance spectrum can predict the fat content of meat samples. For each meat sample, the data consists of a 100-channel spectrum of absorbance records and the levels of moisture (water), fat, and protein. The absorbance is $-\log_{10}$ of the transmittance measured by the spectrometer. The moisture, fat, and protein levels are determined by analytic chemistry. Divide the data randomly into train and test sets (50/50) using the codes from the lectures.

1. Assume that Fat can be modeled as a linear regression in which absorbance characteristics (Channels) are used as features.

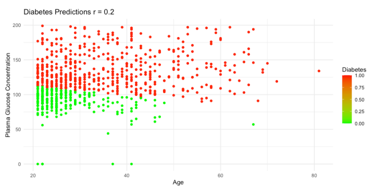
Figure 1: diabetes pred 0.2

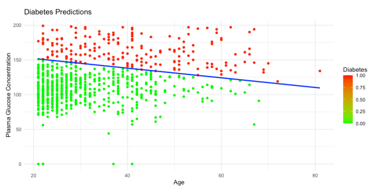
Figure 2: diabetes pred 0.5

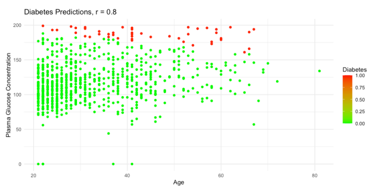
Figure 3: diabetes pred 0.8

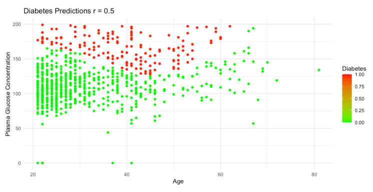
Figure 4: diabetes pred final

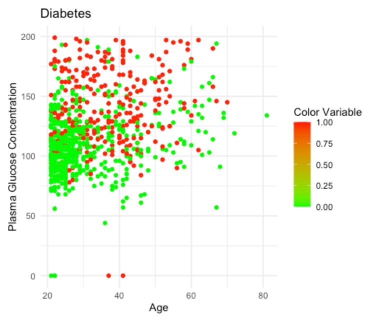

Figure 5: task 1 scatter plot

- Report the underlying probabilistic model.
- Fit the linear regression to the training data and estimate the training and test errors.
- Comment on the quality of fit and prediction and therefore on the quality of the model.

2. Assume now that Fat can be modeled as a LASSO regression in which all Channels are used as features.
   - Report the cost function that should be optimized in this scenario.
3. Fit the LASSO regression model to the training data.
   - Present a plot illustrating how the regression coefficients depend on the log of the penalty factor (log λ).
   - Interpret this plot. What value of the penalty factor can be chosen if we want to select a model with only three features?
4. Repeat step 3 but fit Ridge regression instead of LASSO and compare the plots from steps 3 and 4.
   - Comment on the conclusions.
5. Use cross-validation with the default number of folds to compute the optimal LASSO model.
   - Present a plot showing the dependence of the CV score on log λ and comment on how the CV score changes with log λ.
   - Report the optimal λ and how many variables were chosen in this model.
   - Does the information displayed in the plot suggest that the optimal λ value results in a statistically significantly better prediction than log $\lambda = -4$?
   - Finally, create a scatter plot of the original test versus predicted test values for the model corresponding to the optimal λ and comment on whether the model predictions are good.

```
library(glmnet)

data=read.csv("../tecator.csv", header = TRUE)

# Set random seed so lab result is predictable
set.seed(12345)

# Create variable without protein and moisture columns.
channel_data <-data.frame(data[c(2:102)])

# Split the dataset into training (50%), and test (50%).
n = dim(channel_data)[1]          # Total nr of rows in dataset
id = sample(1:n, floor(n * 0.5))    # Randomly selects 50 % rows for training

id1 = setdiff(1:n, id)          # Remaining rows
data_train = channel_data[id, ] # Assign rows to training set
data_test = channel_data[id1, ] # Set remaining rows as test data

# Task 1
# Create a linear regression model where Fat is the target
# . indicates all other columns are prediction variables
model=lm(Fat~., data=data_train)
model

predict_train <- predict(model,data_train) # make predictions for training data
predict_test <- predict(model,data_test) # make predictions for test data

# calculate difference between predictions and actual values for both data sets
train_diff <- (data_train$Fat - predict_train)
test_diff <- (data_test$Fat - predict_test)

# Caclulate Squared Error for both data sets.
sum(train_diff^2)/dim(data_train)[1]
sum(test_diff^2)/dim(data_test)[1]

# Model is overfitted, probably due to 100 different variables.
# n is barley larger than p
# mse_test = 722.4294
# mse_train = 0.005709117

# Task 3
#Create a matrix without the fat column for the GLMNET model.
X_train_matrix <- as.matrix(data_train[c(1:100)])

# Create the lasso model by setting alpha = 1.
lasso <- glmnet(X_train_matrix, data_train$Fat, alpha = 1, family = "gaussian")

# Look how many features the model has for the lambda = 0.818
coef(lasso, s = 0.818)
plot(lasso, xvar = "lambda")
```

```
# By looking at the graph, a model with only three coefficients is approximately
# log lambda = -0,2 which results in lambda = 0.8187

# Task 4
#Create the ridge model by setting alpha = 0.
ridge <- glmnet(X_train_matrix, data_train$Fat, alpha = 0, family = "gaussian")
plot(ridge,xvar = "lambda")

# task 5
# Train covariane lasso model
cv_lasso=cv.glmnet(X_train_matrix, data_train$Faat, alpha=1 ,family="gaussian")
cv_lasso
cv_lasso$lambda.min
plot(cv_lasso)

coef(cv_lasso, s = cv_lasso$lambda.min)
#8 coefficients are chosen in the model, intercept is NOT included

X_test_matrix <- as.matrix(data_test[c(1:100)])
predictions = predict(cv_lasso, s = "lambda.min", newx = X_test_matrix, type = "response")

sum((predictions - mean(data_test$Fat))^2)/sum((data_test$Fat - mean(data_test$Fat))^2)
sum((predictions - data_test$Fat)^2)

plot(data_test$Fat, col = "black", pch = 16, ylab = "Fat")
points(predictions, col="red", pch = 17)
legend("topleft",inset = c(0, -0.24),
     xpd = TRUE, legend = c("Test data", "Predictions"),
     col = c("black", "red"), pch = c(16, 17)
)
```

5 Lab 2 Task 2: Decision Trees and Logistic Regression for Bank Marketing

The data file bank-full.csv is related to direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact with the same client was required to assess if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Input Variables

- **Bank client data:**
  1. Age (numeric).
  2. Job: type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown').

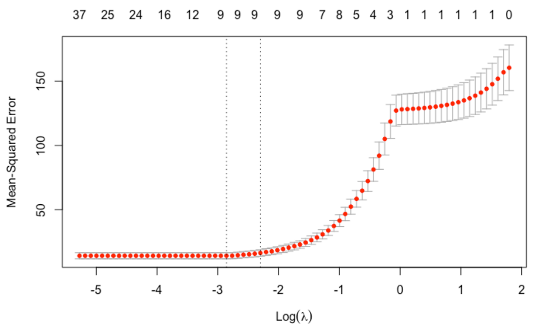

Figure 6: cv lasso

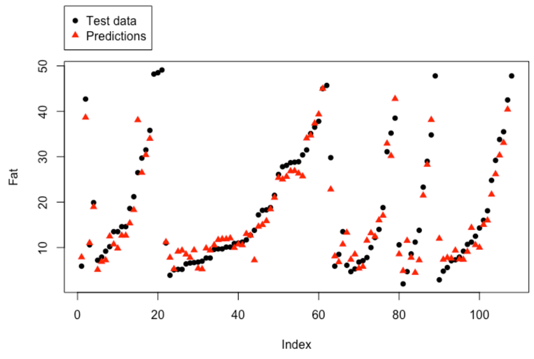

Figure 7: fat scatter

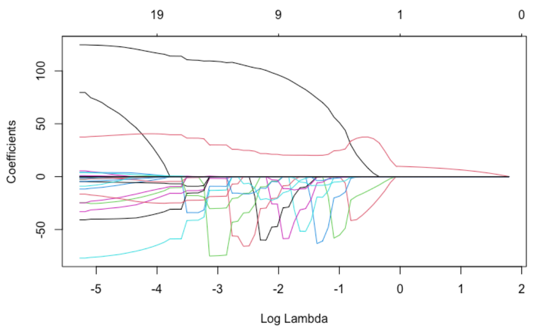

Figure 8: lasso regression

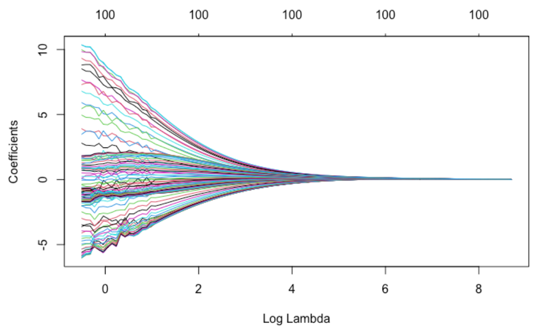

Figure 9: ridge regression plot

3. Marital: marital status (categorical: 'divorced', 'married', 'single', 'unknown'; note: 'divorced' means divorced or widowed).
4. Education (categorical: 'basic.4y', 'basic.6y', 'basic.9y', 'high.school', 'illiterate', 'professional.course', 'university.degree', 'unknown').
5. Default: has credit in default? (categorical: 'no', 'yes', 'unknown').
6. Housing: has housing loan? (categorical: 'no', 'yes', 'unknown').
7. Loan: has personal loan? (categorical: 'no', 'yes', 'unknown').

- **Related to the last contact of the current campaign:**
  1. Contact: contact communication type (categorical: 'cellular', 'telephone').
  2. Month: last contact month of the year (categorical: 'jan', 'feb', ..., 'nov', 'dec').
  3. Day of week: last contact day of the week (categorical: 'mon', 'tue', 'wed', 'thu', 'fri').
  4. Duration: last contact duration, in seconds (numeric). **Important note:** This attribute highly affects the output target (e.g., if duration=0, then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call, y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.
- **Other attributes:**
  1. Campaign: number of contacts performed during this campaign and for this client (numeric, includes the last contact).

2. Pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted).
3. Previous: number of contacts performed before this campaign and for this client (numeric).
4. Poutcome: outcome of the previous marketing campaign (categorical: 'failure', 'nonexistent', 'success').

Output Variable (Target)

- y: Has the client subscribed a term deposit? (binary: 'yes', 'no').

Tasks

1. Import the data to R, remove the variable duration, and divide it into training/validation/test sets as $40/30/30$. Use data partitioning code specified in Lecture 2a.
2. Fit decision trees to the training data by changing the default settings one by one (i.e., not simultaneously):
   a) Decision tree with default settings.
   b) Decision tree with smallest allowed node size equal to 7000.
   c) Decision tree with minimum deviance equal to 0.0005.

   Report the misclassification rates for the training and validation data. Which model is the best among these three? Report how changing the deviance and node size affected the size of the trees and explain why.
3. Use the training and validation sets to choose the optimal tree depth in model 2c. Study the trees up to 50 leaves. Present a graph of the dependence of deviances for the training and validation data on the number of leaves. Interpret this graph in terms of the bias-variance tradeoff. Report the optimal number of leaves and identify which variables seem to be most important for decision-making in this tree. Interpret the information provided by the tree structure.
4. Estimate the confusion matrix, accuracy, and F1 score for the test data by using the optimal model from step 3. Comment on whether the model has good predictive power and which of the measures (accuracy or F1 score) should be preferred here.
5. Perform a decision tree classification of the test data using the following loss matrix:

   $$L = \begin{bmatrix} 0 & 5 \\ 1 & 0 \end{bmatrix}$$

   Report the confusion matrix for the test data. Compare the results with the results from step 4 and discuss how the rates have changed and why.
6. Use the optimal tree and a logistic regression model to classify the test data using the following principle:

   $$Y = \begin{cases} y_{pos} & \text{if } P(Y = y_{pos}|X) > \pi, \\ y_{neg} & \text{otherwise.} \end{cases}$$

   where $\pi = 0.05, 0.1, 0.15, \ldots, 0.9, 0.95$. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. What are your conclusions? Why might the precision-recall curve be a better option here?

```
 ############################ Packages ############################

library(tree)
library(rpart)

############################ PART 1 ############################

# Import bank marketing campaign csv file with str as factors
data_frame = read.csv2("../bank-full.csv", stringsAsFactors = TRUE)

# Remove column "duration" using %in% operator
data_frame = data_frame[ , !names(data_frame) %in% c("duration")]

# Split the dataset into training (50%), validation (25%), and test (25%) sets
n = dim(data_frame)[1]             # Total nr of rows in dataset
set.seed(12345)
id = sample(1:n, floor(n * 0.4))    # Randomly selects 40 % rows for training
data_train = data_frame[id, ]       # Assign rows to training set

id1 = setdiff(1:n, id)            # Remaining rows
set.seed(12345)
id2 = sample(id1, floor(n * 0.30))  # Randomly selects 30 % rows for validation
data_validation = data_frame[id2, ] # Assign rows to validation set

id3 = setdiff(id1, id2)           # Remaining rows
data_test = data_frame[id3, ]       # Assign rest of 30 % rows to testing


############################ PART 2 ############################

# Fit decision trees to training dataset
tree_a = tree(formula = y ~ ., data = data_train)
tree_b = tree(formula = y ~ ., data = data_train, control = tree.control(nrow(data_train),
↪   minsize = 7000))
tree_c = tree(formula = y ~ ., data = data_train, control = tree.control(nrow(data_train),
↪   mindev = 0.0005))

# Predictions on validation dataset
predictions_a = predict(tree_a, newdata = data_validation, type = "class")
predictions_b = predict(tree_b, newdata = data_validation, type = "class")
predictions_c = predict(tree_c, newdata = data_validation, type = "class")

# Misclassification rates for the training dataset
summary(tree_a) # Misclassification error rate: 0.1048
summary(tree_b) # Misclassification error rate: 0.1048
summary(tree_c) # Misclassification error rate: 0.09362

# Misclassification rates for the validation dataset
misclass = function(X, X1) {
  n = length(X)
  return(1 - sum(diag(table(X, X1))) / n)
}
misclass(predictions_a, data_validation$y) # Misclassification error rate: 0.1092679
misclass(predictions_b, data_validation$y) # Misclassification error rate: 0.1092679
misclass(predictions_c, data_validation$y) # Misclassification error rate: 0.1118484

#The best models are A and B, as they have low misclassification rates for both the training
↪    set and the validation set. The problem with Model C is that it has a good
↪    misclassification rate for the training set but a higher misclassification rate for the
↪    validation set, which is typical of overfitting.

#A larger minimum node size increases the model misclassification rate while a lower does
↪    not improve performance. Increasing the minsize value forces the tree to stop splitting
↪    when a node's number of obersvations falls below the threshold. This results in simpler
↪    and more general trees. Lowering the minsize can allow the tree to grow very deep,
↪    increasing the risk of overfitting. While high values the tree is constrained from splitting
↪    too much, which can lead to underfitting.

#A larger minimum deviance makes the model better for the validation set but worse for the
↪    training set. This is because a higher minimum deviance results in a smaller and simpler
↪    tree, reducing the risk of overfitting but increasing the risk of underfitting. Lowering the
↪    minimum deviance allows the tree to grow larger and more complex, improving
↪    performance on the training set but increasing the risk of overfitting, which can lead to
↪    poor generalization and worse performance on the validation set.

########################### PART 3 ###########################

# Init vector with length 50 to store score
train_score = rep(0, 50)
valid_score = rep(0, 50)

# Prune the C tree starting with at least 2 node leaves up to 50
# Calculate the deviance on both datasets and save to the score vector.
for(leaves in 2:50) {
  # Prune Tree C to current "leaves" number
  pruned_tree = prune.tree(tree_c, best = leaves)

  # Predictions on validation dataset using pruned tree
  predictions_valid = predict(pruned_tree, newdata = data_validation, type = "tree")

  train_score[leaves] = deviance(pruned_tree)
  valid_score[leaves] = deviance(predictions_valid)
}

# Graph of the dependence of deviance for the datasets on the number of leaves
plot(2:50, train_score[2:50], type = "b", col = "red", ylim = c(min(valid_score[2:50]),
↪    max(train_score[2:50])),
     main = "Optimal tree depth", ylab = "Deviance", xlab = "Number of leaves")
points(2:50, valid_score[2:50], type = "b", col = "blue")
legend("topright", c("train data", "validation data"), fill = c("red", "blue"))

# Optimal number of leaves that minimize training deviance.
optimal_train = which.min(train_score[2:50])
```

```
optimal_validation = which.min(valid_score[2:50])

optimal_tree = prune.tree(tree_c, best = optimal_validation)

# Display optimal tree
plot(optimal_tree)
text(optimal_tree, pretty=0)
optimal_tree
summary(optimal_tree)

# The optimal number of leaves is 47 for the training dataset and 21 for the validation
↪   dataset. The bias-variance tradeoff explains how a model's complexity influences
↪   prediction accuracy and generalization to unseen data. A tree with many leaves (greater
↪   depth) has low bias but high variance, potentially leading to overfitting. A tree with fewer
↪   leaves has low variance but high bias, which can result in underfitting. As model
↪   complexity increases, variance rises while bias decreases. In decision trees, the number
↪   of leaves directly impacts this balance as it defines the complexity of the tree. The key is
↪   to find the optimum where there is a balance between bias and variance so the error is
↪   minimized, see Figure \ref{fig:task_3_graph}.

# The variables which are the most important is poutcome which is the root node but also
↪   month, contact and pdays which appear frequently in splits in the tree, see Figure
↪   \ref{fig:task3_optimal_tree}.

############################ PART 4 ############################

# Predictions on the test dataset using optimal tree
predications_test = predict(optimal_tree, newdata = data_test, type = "class")

# Creating confusion matrix
conf_matrix = table(data_test$y, predications_test)
conf_matrix

# Calculates the accuracy and f1 score from confusion matrix
accuracy_and_f1 = function(conf_matrix) {
  TP = conf_matrix["yes", "yes"] # True positives
  TN = conf_matrix["no", "no"] # True negatives
  FP = conf_matrix["no", "yes"] # False positives
  FN = conf_matrix["yes", "no"] # False negatives

  precision = TP / (TP + FP)
  recall = TP / (TP + FN)
  accuracy = (TN + TP) / (TN + FP + TP + FN)
  f1_score = 2 * (precision * recall) / (precision + recall)

  return(c(accuracy = accuracy, f1_score = f1_score))
}

# Print result
result = accuracy_and_f1(conf_matrix)
print(result["accuracy"])
print(result["f1_score"])
```

```r
# The model has an accuracy of 0.8923 and an F1-score of 0.2849. The high accuracy might
↪   seem good, but it’s mainly due to the imbalance in the dataset. The F1-score, which
↪   considers both precision and recall, gives a better picture and shows that the model
↪   struggles to predict the less common class accurately. This shows why accuracy alone
↪   isn’t a reliable metric for imbalanced datasets.


########################### PART 5 ###########################

# Loss matrix defined in task
loss_matrix = matrix(c(0, 1, 5, 0), byrow = TRUE, nrow = 2,
                     dimnames = list(c("no", "yes"), c("no", "yes")))

# Predictions on the test dataset using optimal tree
predications_test = predict(optimal_tree, newdata = data_test)

# Predictions multiplied with loss matrix
losses = predications_test %*% loss_matrix

# For each observation (row) it finds the class (column)
# with the smallest loss and returns the index of that class.
predicted_classes = apply(losses, 1, which.min)

# Map the correct column names (no, yes) for the predicted classes
predicted_classes = colnames(losses)[predicted_classes]

# Create confusion matrix
conf_matrix = table(data_test$y, predicted_classes)
conf_matrix

# Print result
result = accuracy_and_f1(conf_matrix)
print(result["accuracy"])
print(result["f1_score"])

# The model accuracy has decreased to 0.8732, while the F1-score has improved to 0.4826.
↪   This change is due to the introduction of a loss matrix that imposes a heavier penalty on
↪   false negatives compared to false positives. Correct predictions, represented by the
↪   diagonal entries of the matrix, are not penalized.

########################### PART 6 ###########################

tpr_and_fpr = function(conf_matrix) {
  TP = conf_matrix["yes", "yes"] # True positives
  TN = conf_matrix["no", "no"] # True negatives
  FP = conf_matrix["no", "yes"] # False positives
  FN = conf_matrix["yes", "no"] # False negatives

  tpr = TP / (TP + FN) # true positive rate
  fpr = FP / (FP + TN) # false positive rate
  return(c(tpr = tpr, fpr = fpr))
}

# True positive rates and False positive rates
tpr_tree = c()
fpr_tree = c()
tpr_lrm = c()
fpr_lrm = c()

# Logistic regression model
lrm = glm(formula = y ~ ., data = data_train, family = "binomial")

# Predictions on test data with both models
predictions_test = predict(optimal_tree, newdata = data_test, type = "vector")
predictions_lrm = predict(lrm, newdata = data_test, type = "response")

for (pi in seq(from = 0.05, to = 0.95, by = 0.05)) {
  # Optimal tree predictions
  y_hat_optimal_tree = ifelse(predictions_test[, "yes"] > pi, "yes", "no")
  conf_matrix = table(data_test$y, factor(y_hat_optimal_tree, levels = c("no", "yes")))
  result = tpr_and_fpr(conf_matrix)
  tpr_tree = append(tpr_tree, result["tpr"])
  fpr_tree = append(fpr_tree, result["fpr"])

  # Logistic regression model
  y_hat_lrm = ifelse(predictions_lrm > pi, "yes", "no")
  conf_matrix = table(data_test$y, y_hat_lrm)
  result = tpr_and_fpr(conf_matrix)
  tpr_lrm = append(tpr_lrm, result["tpr"])
  fpr_lrm = append(fpr_lrm, result["fpr"])
}

# Plot ROC curves
plot(fpr_tree, tpr_tree, type = "b", col = "red", ylim = c(min(tpr_tree), max(tpr_tree))
     ,xlim = c(min(fpr_tree), max(fpr_tree)),
     main = "ROC curves", ylab = "True positive rate", xlab = "False positive rate")
points(fpr_lrm, tpr_lrm, type = "b", col = "blue")
legend("bottomright", c("Optimal tree", "Logistic regression model"), fill = c("red", "blue"))

# The ROC curve in Figure \ref{fig:roc_curves} shows that the optimal tree model performs
↪   slightly better than the logistic regression model, with higher True Positive Rate (TPR)
↪   across the False positive Rate (FPR) values. Since the test data has very imbalanced
↪   classes, the precision-recall curve would be a better choice here. Because it focuses on
↪   the minority class, which would be more informative for an imbalanced dataset such as
↪   this.
```

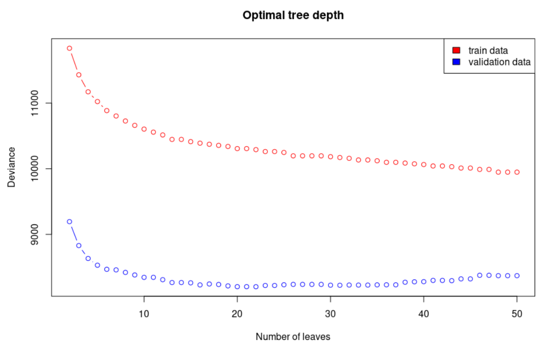

Figure 10: dependency deviation

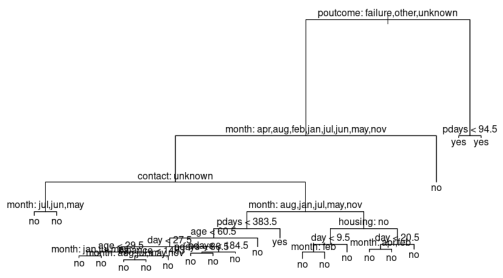

Figure 11: optimal tree

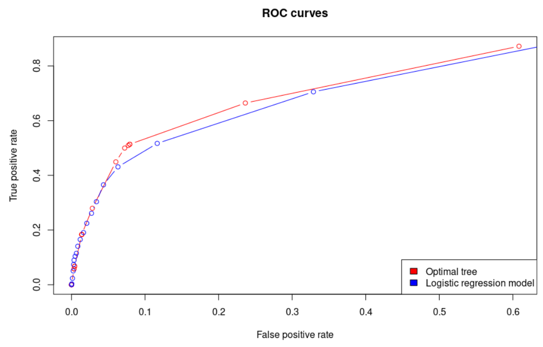

Figure 12: roc curves

6 Lab 2 Task 3: Principal Components and Implicit Regularization

The data file communities.csv contains the results of studies of the crime level in the United States based on various characteristics of a given location. The main variable studied is ViolentCrimesPerPop, which represents the total number of violent crimes per 100K population.

Tasks

1. Scale all variables except ViolentCrimesPerPop and implement PCA using the eigen() function.
    - Report how many components are needed to obtain at least 95% of the variance in the data.
    - What is the proportion of variation explained by each of the first two principal components?
2. Repeat the PCA analysis using the princomp() function and make the trace plot of the first principal component.
    - Do many features have a notable contribution to this component?
    - Report which 5 features contribute the most (by absolute value) to the first principal component.

    - Comment on whether these features have anything in common and whether they may have a logical relationship to the crime level.
    - Provide a plot of the PC scores in the coordinates (PC1, PC2) where the color of the points corresponds to ViolentCrimesPerPop. Analyze this plot (hint: use the ggplot2 package).
3. Split the original data into training and test sets (50/50), scale both features and response appropriately, and estimate a linear regression model from the training data where ViolentCrimesPerPop is the target and all other data columns are features.
    - Compute the training and test errors for these data.
    - Comment on the quality of the model.
4. Implement a function that depends on the parameter vector θ and represents the cost function for linear regression without intercept on the training dataset.
    - Use the BFGS method (optim() function without the gradient specified) to optimize this cost with the starting point $\theta_0 = 0$.
    - Compute the training and test errors for every iteration number.
    - Present a plot showing the dependence of both errors on the iteration number.
    - Comment on which iteration number is optimal according to the early stopping criterion.
    - Compute the training and test errors in the optimal model, compare them with the results in step 3, and make conclusions.

Hints

- **Hint 1:** Don't store parameters from each iteration (this will require a lot of memory). Instead, compute and store test errors directly.
- **Hint 2:** Discard some initial iterations (e.g., 500) in your plot to make the dependencies visible.

```
data <- read.csv('communities.csv')

# task 1

library(caret)
scaler<-preProcess(data)
dataS<-predict(scaler,data)
dataS$ViolentCrimesPerPop <- data$ViolentCrimesPerPop # no scaling on
↪   ViolentCrimesPerPop according to instructions
S <- (1/101) * t(as.matrix(dataS)) %*% as.matrix(dataS) # covariance matrix, how variables
↪   depends on each other

eig <- eigen(S)
lambda <- eig$values
lambda_perc <- lambda/sum(lambda)*100 # gives the variance of each 'projection' (how
↪   much data the projection can represent)
sprintf("%2.3f",lambda_perc)
# prints:
# [1]  "25.025" "16.931" "9.298"  "7.554"  "5.660"  "4.236"  "3.226"  "2.969"  "2.067"  "1.617"
↪   "1.572"  "1.470"
```

```
# [13] "1.414"  "1.030"  "0.931"  "0.890"  "0.748"  "0.705"  "0.649"  "0.638"  "0.624"  "0.568"
↪  "0.542"  "0.519"
# [25] "0.504"  "0.480"  "0.466"  "0.451"  "0.431"  "0.386"  "0.365"  "0.351"  "0.337"  "0.311"
↪  "0.287"  "0.259"
# [37] "0.256"  "0.245"  "0.240"  "0.222"  "0.211"  "0.205"  "0.200"  "0.190"  "0.183"  "0.165"
↪  "0.160"  "0.142"
# [49] "0.138"  "0.126"  "0.112"  "0.107"  "0.104"  "0.100"  "0.090"  "0.081"  "0.077"  "0.073"
↪  "0.069"  "0.066"
# [61] "0.064"  "0.062"  "0.058"  "0.051"  "0.049"  "0.046"  "0.044"  "0.042"  "0.040"  "0.038"
↪  "0.035"  "0.034"
# [73] "0.032"  "0.031"  "0.028"  "0.026"  "0.024"  "0.022"  "0.021"  "0.020"  "0.018"  "0.018"
↪  "0.016"  "0.015"
# [85] "0.014"  "0.013"  "0.011"  "0.009"  "0.008"  "0.006"  "0.006"  "0.005"  "0.004"  "0.004"
↪  "0.003"  "0.002"
# [97] "0.002"  "0.001"  "0.001"  "0.001"  "0.001"
sum(lambda_perc[1:34]) # 94.9
sum(lambda_perc[1:35]) # 95.2 => we need 35 variables for a variance of 95 %

# task 2

res <- princomp(dataS)
lambda <- res$sdev^2 # variance = standard_deviation^2
sprintf("%2.3f",lambda/sum(lambda)*100) # prints as above
U1<- (res$loadings)[,1]
plot(sort(abs(U1)), main="Traceplot, PC1")
top_5 <- tail(sort(abs(U1)),5)
top_5
# PctPopUnderPov    pctWInvInc   PctKids2Par     medIncome     medFamInc
# 0.1737183         0.1748076    0.1755406       0.1818171     0.1831478
# It seems that poverty is the largest contributing factor

library(ggplot2)
S1 <- (res$scores)[,1]
S2 <- (res$scores)[,2]

ggplot() + geom_point(aes(x=S1, y=S2, color=dataS$ViolentCrimesPerPop)) +
  labs(title = "PC1 and PC2 scores",
       x = "PC1",
       y = "PC2") +
  theme_minimal()

# TASK 3

n<-dim(data)[1]
set.seed(12345)
id<-sample(1:n, floor(n*0.5))
train<-data[id,]
test<-data[-id,]

scaler<-preProcess(train)
trainS<-predict(scaler,train)
testS<-predict(scaler,test)
```

```
fit<-lm(ViolentCrimesPerPop ~ ., data=trainS)
summary <- summary(fit)

dim(summary$coefficients)

MSE_train <- mean(residuals(fit)^2) # MSE for train = 0.2752071
MSE_test <- mean((test$ViolentCrimesPerPop - predict(fit, newdata=testS))^2) # MSE for
↪   test = 0.5408757

# We see that the MSE is quite good for train but quite high for test meaning it is alright but
↪   not more.
# Slightly overfitted
# through the summary(fit) we see that racepctblack, PctWorkMom, PctPersDenseHous are
↪   significant contributors (***)

# Task 4 #########

# cost function for linear regression w/o intercept
trainS_X <- as.matrix(subset(trainS, select=-ViolentCrimesPerPop))
testS_X <- as.matrix(subset(testS, select=-ViolentCrimesPerPop))
trainS_Y <- trainS$ViolentCrimesPerPop
testS_Y <- testS$ViolentCrimesPerPop

train_MSE_vec <- c()
test_MSE_vec <- c()

MSE <- function(theta)
{
  MSE_train <- mean( (trainS$ViolentCrimesPerPop - trainS_X %*% theta)^2 )
  MSE_test <- mean( ( testS_Y - testS_X %*% theta )^2 )

  train_MSE_vec <<- c(train_MSE_vec, MSE_train)
  test_MSE_vec <<- c(test_MSE_vec, MSE_test)

  return(MSE_train)
}

optim(rep(0, 100), fn = MSE, method = "BFGS")

interval <- 1:length(train_MSE_vec)
ggplot() +
  geom_point(aes(interval, train_MSE_vec[interval]), color="red") +
  geom_point(aes(interval, test_MSE_vec[interval]), color="blue") +
  ylim(0,1) +
  labs(title = "MSE train (red) and test (blue)", x = "Iteration", y = "MSE Score")

# early stopping criterion says to stop at the lowest test MSE
optimal_MSE_train <- train_MSE_vec[which.min(test_MSE_vec)] # = 0.3032999
optimal_MSE_test <- min(test_MSE_vec) # = 0.4002329
# We see that this is a slight elevation from the previous MSE of 0.275 for test
# but a fairly good decrease from 0.541 for the test MSE.
# Overall this is probably a better model
```

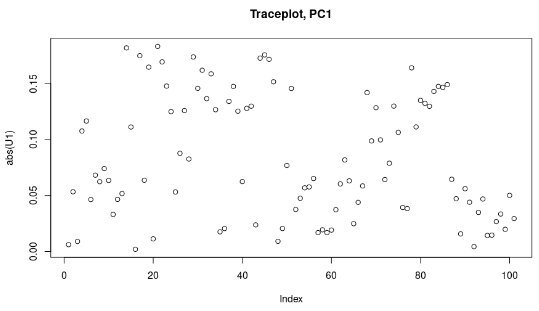

Figure 13: Traceplot PC1

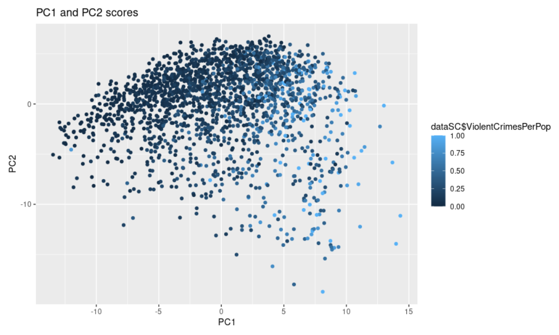

Figure 14: PC2 (y-axis) vs PC1 (x-axis)

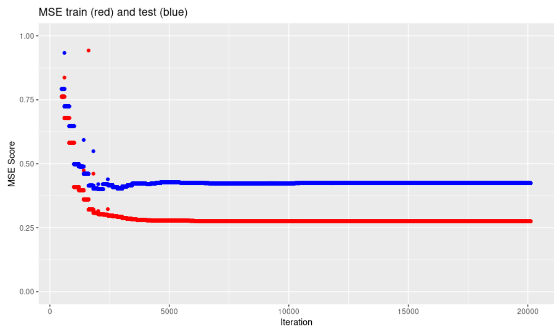

Figure 15: task 4: train = red, test = blue

7 Lab 3 Task 1: Kernel Method for Predicting Hourly Temperatures in Sweden

You are tasked with implementing a kernel method to predict hourly temperatures for a specific date and location in Sweden using the provided files stations.csv and temps50k.csv. These files contain data on weather stations and temperature measurements from the Swedish Meteorological and Hydrological Institute (SMHI).

Objective

Provide a temperature forecast for a given date and location in Sweden. The forecast should include predicted temperatures from 4 am to midnight at 2-hour intervals.

Methodology

Use a kernel composed of three Gaussian kernels:

1. **Physical Distance Kernel:** Accounts for the physical distance between a station and the point of interest. Use the distHaversine function from the geosphere package.

2. **Date Distance Kernel:** Accounts for the temporal distance between the measurement date and the prediction date.
3. **Time Distance Kernel:** Accounts for the temporal distance between the measurement time and the prediction time.

Choose appropriate smoothing coefficients ($h_{\text{distance}}, h_{\text{date}}, h_{\text{time}}$) for each kernel manually. Show kernel values as a function of distance to demonstrate how closer points have higher values.

Steps

1. Filter out temperature measurements that are posterior to the prediction date and time.
2. Compute kernel values for the physical distance, date distance, and time distance.
3. Combine the three kernels:
   - First, by summing them.
   - Then, by multiplying them.
4. Generate forecasts for temperatures at the specified times.
5. Compare results obtained using the summation and multiplication methods and explain differences.

Template Code

Below is the R template code:

```r
set.seed(1234567890)
library(geosphere)

# Load data
stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations, temps, by = "station_number")

# Smoothing coefficients (to be determined manually)
h_distance <- # Physical distance smoothing coefficient
h_date <-     # Date smoothing coefficient
h_time <-     # Time smoothing coefficient

# Prediction parameters
a <- 58.4274  # Latitude of the point to predict
b <- 14.826   # Longitude of the point to predict
date <- "2013-11-04" # Date to predict
times <- c("04:00:00", "06:00:00", "08:00:00", ..., "24:00:00") # Times

# Initialize temperature predictions
temp <- vector(length = length(times))

# Kernel implementation and prediction (students' code here)

# Visualization
```

```r
plot(temp, type = "o", xlab = "Time", ylab = "Temperature (C)",
     main = "Predicted Temperatures")
```

Analysis

After implementing the above, repeat the exercise by combining the kernels through multiplication instead of summation. Discuss the differences in results and provide insights into why the two methods yield different outcomes.

```r
# setwd(paste0(getwd(),"/tdde01-machine-learning/lab3/"))

set.seed(1234567890)
library(geosphere)

stations <- read.csv("stations.csv", fileEncoding = "latin1")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")

h_pos <- c(58.41086,15.62157) # Linkoping's latitude and longitude
h_time <- "15:10:22"
h_date <- "2013-11-04"
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00",
           "14:00:00", "16:00:00", "18:00:00", "20:00:00", "22:00:00",
           "24:00:00")

##### functions ##################################################

## finding the width of the first kernel
# max_lat_long <- c( max(st$latitude), max(st$longitude) )
# min_lat_long <- c( min(st$latitude), min(st$longitude) )
# max_dist = distHaversine(
#   max_lat_long,
#   min_lat_long
#   )
# l <- 6378137 / 10 # earth radius / 10
# exp( - ( max_dist^2 ) / ( 2 * l^2 ) ) # gives practically zero which is good

earth_distance <- c()
gaussian_kernel_earth <- function(x, data)
{
  x_prim <- matrix( c(data$latitude, data$longitude), nrow = length(data$longitude), ncol=2)

  l = 6378137 / 10 # standard deviation or in this case width = earth radius / 10
  distance <- c()
  for (i in 1:dim(x_prim)[1])
  {
    distance <- c(distance, distHaversine(x, x_prim[i,]))
  }
  earth_distance <<- distance
  exp( - ( ( distance  )^2 ) / ( 2 * l^2 ) )
}

#
↪ ----------------------------------------------------------------------------------

days_distance <- c()
gaussian_kernel_days <- function(x, data)
{
  x_prim <- data$date
  l = sqrt(365 / 2) # standard deviation in gaussian kernel = sqrt(variance)
  x <- rep( x, length(x_prim) )
  x_prim <- x_prim

  days_diff <- as.numeric( difftime(x, x_prim) ) %% 365
  days_distance <<- days_diff

  exp( - ( ( days_diff  )^2 ) / ( 2 * l^2 ) )
}

#
↪ ----------------------------------------------------------------------------------

hours_distance <- c()
gaussian_kernel_hours <- function(x, data)
{
  x_prim <- as.POSIXct(data$time, format = "%H:%M:%S" )
  x <- as.POSIXct(rep(x, length(data$time)) , format = "%H:%M:%S" )
  l = sqrt(24 / 2) # standard deviation in gaussian kernel = sqrt(variance)
  time_diff <- abs( as.numeric( difftime(x,x_prim, units="hours") ) ) %% 24
  hours_distance <<- time_diff
  exp( - ( ( time_diff^2 ) / ( 2 * l^2 ) ) )
}

delete_posterior_dates <- function(current_date, data)
{
  delete_index <- c()
  for (i in 1:dim(data)[1])
  {
    if (difftime(current_date, data$date[i]) <= 0)
    {
      delete_index <- c(delete_index, i)
    }
  }
  data[-delete_index,]
}

# delete_posterior_dates("1960-04-20", st)

# further tweaking of the width (l)
# dist_kernel <- gaussian_kernel_earth( h_pos, st )
# mean( dist_kernel )
# max( dist_kernel )
# min( dist_kernel )
#
# days_kernel <- gaussian_kernel_days(h_date, st)
# mean(days_kernel)
# max(days_kernel)
# min(days_kernel)
# #
# # hours_kernel <- gaussian_kernel_hours(times[1], st)
# mean(hours_kernel)
# max(hours_kernel)
# min(hours_kernel)

total_kernel <- function(pos, date, times, data, mult_add)
{
  data <- delete_posterior_dates(date, data)

  dist_kernel <- gaussian_kernel_earth( pos, data )
  days_kernel <- gaussian_kernel_days( date, data )

  total_kernel_vector <- c()
  for (i in 1:length(times))
  {
    time_kernel <- gaussian_kernel_hours( times[i], data )
    # print(time_kernel)
    if (mult_add == "add")
      total_kernel_vector <- c(total_kernel_vector, dist_kernel + days_kernel + time_kernel)
    else
      total_kernel_vector <- c(total_kernel_vector, dist_kernel * days_kernel * time_kernel)
  }
  total_matrix <- matrix(total_kernel_vector, nrow = dim(data)[1], ncol = length(times))

  temp_pred <- c()
  air_temperatures <- data$air_temperature
  for (i in 1:length(times))
  {
    # Nadaraya-Watson weighted average, eq 2.41, page 35, course book
    temp_pred <- c(temp_pred, (air_temperatures %*% total_matrix[,i]) / sum(total_matrix[,i]) )
  }
  temp_pred
}

############################################################

earth_kernel <- gaussian_kernel_earth( h_pos, st )
days_kernel <- gaussian_kernel_days( h_date, st )
hours_kernel <- gaussian_kernel_hours( times[1], st )

plot( earth_distance, earth_kernel, main = "earth")
plot( days_distance, days_kernel, main = "days" )
plot( hours_distance,  hours_kernel, main = "hours")
```

```
time_of_day <- c()
for (i in 1:length(times))
{
  time_of_day <- c(time_of_day, (strsplit(times[i], split = ":"))[[1]][1] )
}
time_of_day <- as.numeric(time_of_day)
times <- c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00",
           "14:00:00", "16:00:00", "18:00:00", "20:00:00", "22:00:00",
           "24:00:00")
pos <- c(58.41086,15.62157)

date <- "2010-06-21"
temp <- total_kernel(pos, date, times, st, "add")
plot(time_of_day, temp, type="o", main=date)
print(date)

date <- "2010-12-24"
temp <- total_kernel(pos, date, times, st, "add")
plot(time_of_day, temp, type="o", main=date)
print(date)

date <- "2010-06-21"
temp <- total_kernel(pos, date, times, st, "mult")
plot(time_of_day, temp, type="o", main=date)
print(date)

date <- "2010-12-24"
temp <- total_kernel(pos, date, times, st, "mult")
plot(time_of_day, temp, type="o", main=date)
print(date)

# We see that the sum kernel seems to always give values around 4 degree celsius
# whereas the multiplication kernel seems to give more accurate values.
# This is thought to be because multiplying values between 0 and 1 really accentuates
# the 'strong' values close to one and it really diminishes the values close to 0,
# whereas when summing the values it is possible that two values even each other out
# making it essentially only depend on the last value which might not contribute majorly
# to the result.
```

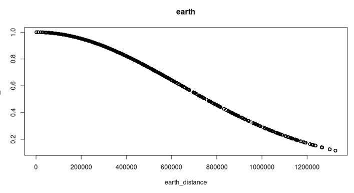

Figure 16: geo kernel

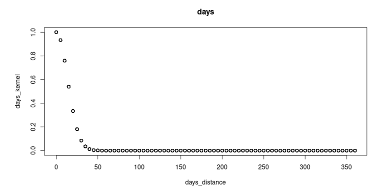

Figure 17: days kernel

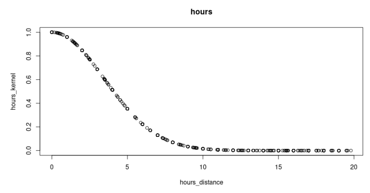

Figure 18: hours kernel

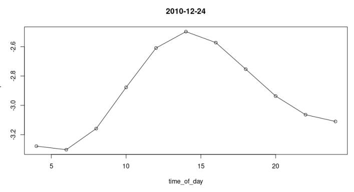

Figure 19: winter mult

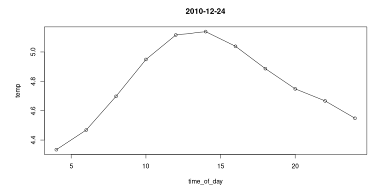

Figure 20: winter sum

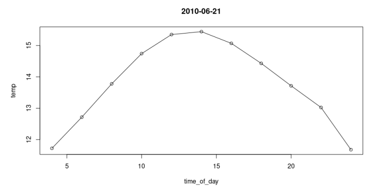

Figure 21: summer mult

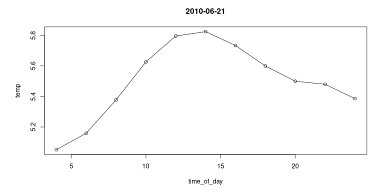

Figure 22: summer sum

8 Lab 3 Task 2 SVM Model Selection for Spam Dataset

The code in the file Lab3Block1 2021 SVMs St.R performs SVM model selection to classify the spam dataset. To do so, the code uses the function ksvm from the R package kernlab, which also includes the spam dataset. All the SVM models to select from use the radial basis function kernel (also known as Gaussian) with a width of 0.05. The C parameter varies between the models.

Run the code in the file Lab3Block1 2021 SVMs St.R and answer the following questions:

1. Which filter do you return to the user? filter0, filter1, filter2 or filter3? Why?
2. What is the estimate of the generalization error of the filter returned to the user? err0, err1, err2 or err3? Why?
3. Once an SVM has been fitted to the training data, a new point is essentially classified according to the sign of a linear combination of the kernel function values between the support vectors and the new point. You are asked to implement this linear combination for filter3.
   - You should make use of the functions alphaindex, coef, and b that return the indexes of the support vectors, the linear coefficients for the support vectors, and the negative intercept of the linear combination.
   - See the help file of the kernlab package for more information.
   - You can check if your results are correct by comparing them with the output of the function predict where you set type = "decision".
   - Do so for the first 10 points in the spam dataset.

- Feel free to use the template provided in the file Lab3Block1 2021 SVMs St.R.

```
library(kernlab)
set.seed(1234567890)

data(spam)
foo <- sample(nrow(spam))
spam <- spam[foo,]
tr <- spam[1:3000, ]
va <- spam[3001:3800, ]
trva <- spam[1:3800, ]
te <- spam[3801:4601, ]
by <- 0.3
err_va <- NULL
for(i in seq(by,5,by)){
  filter <- ksvm(type~.,data=tr,kernel="rbfdot",
  kpar=list(sigma=0.05),C=i,scaled=FALSE)
  mailtype <- predict(filter,va[,-58])
  t <- table(mailtype,va[,58])
  err_va <-c(err_va,(t[1,2]+t[2,1])/sum(t))
}
filter0 <- ksvm(type~.,data=tr,kernel="rbfdot",
kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
mailtype <- predict(filter0,va[,-58])
t <- table(mailtype,va[,58])
err0 <- (t[1,2]+t[2,1])/sum(t)
err0
filter1 <- ksvm(type~.,data=tr,kernel="rbfdot",
kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
mailtype <- predict(filter1,te[,-58])
t <- table(mailtype,te[,58])
err1 <- (t[1,2]+t[2,1])/sum(t)
err1
filter2 <- ksvm(type~.,data=trva,kernel="rbfdot",
kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
mailtype <- predict(filter2,te[,-58])
t <- table(mailtype,te[,58])
err2 <- (t[1,2]+t[2,1])/sum(t)
err2
filter3 <- ksvm(type~.,data=spam,kernel="rbfdot",
kpar=list(sigma=0.05),C=which.min(err_va)*by,scaled=FALSE)
mailtype <- predict(filter3,te[,-58])
t <- table(mailtype,te[,58])
err3 <- (t[1,2]+t[2,1])/sum(t)
err3

# Questions

# 1. Which filter do we return to the user ? filter0, filter1, filter2 or filter3? Why?
# 2. What is the estimate of the generalization error of the filter returned to the user? err0,
↪   err1, err2 or err3? Why?
# 3. Implementation of SVM predictions.
```

```
sv<-alphaindex(filter3)[[1]]
co<-coef(filter3)[[1]]
inte<- - b(filter3)

k<-NULL
for(i in 1:10){ # We produce predictions for just the first 10 points in the dataset.
  k2<-NULL
  # Calculate k_values for each support vector.
  for(j in 1:length(sv)){
    # Calculate the difference between point and current support vector point
    diff <- spam[i, -58] - spam[sv[j],-58]
    # Calculate the value of the gaussian kernel for the current points
    k_val<- exp(-0.05 * sum(diff^2))
    # Add to vector of all k_values of the current point
    k2 <- c(k2, k_val)
  }
  # Calculate decision function for the current point
  k <- c(k, sum(co * k2) + inte)
}
print(k)
print(spam[1:10, 58])
predict(filter3,spam[1:10,-58], type = "decision")
```

Answers:
1.

- Filter0: trained on the training data, tested on validation data.
- Filter1: trained on the training data, tested on test data.
- Filter2: trained on training + validation data, tested on test data.
- Filter3: trained on the entire dataset, tested on test data.

A model performs better if it is trained on more data. As long as that data is accurate. Filter 0 and 1 are trained on only the training set, but we want our model to be trained on as much data as possible. Filter 2 is trained on training + validation data. However, filter3 is trained on the entirety of the spam data set. Therefore it's the most appropriate filter to return to someone who needs to use a model for making predictions on new data.
2.

- err0: obatained by filter0 which is trained on training data and validated on validation data. not appropriate becasue not trained on validation data. Validation data was used to choose best C. So it has already "seen" some of the data on which it was tested.
- err1: obatained by filter1 which is trained on training data and validated on tested data. Not appropriate becasue not trained on validation data, I.E not the most amount of data possible which isn't test data.
- err2: obatained by filter2 which is trained on training + validation and test data. Thus an appropriate generalization error because the model is trained on the largest possible amount of data without having "seen" any of the test data.

- err3: obatained by filter3 which is trained on all data and validated on test data. Not appropriate because it has ”seen” the test data.

Based on the reasoning above, err2 is the estimate of the generalization error of the filter returned to the user. The other errors are validated on the wrong dataset, are not trained on a satisfiable amount of data or have been trained on the data on which it is being validated.
3.
The decision function for the SVM for a point x_* is the following.

$$\hat{y}(x_*) = \text{sign}(\hat{\alpha}^\top K(X, x_*) + \beta)$$

Where $\hat{\alpha}^\top$ is a vector where non-zero values represent the support vectors. $K(X, x_*))$ is a vector created by applying the kernel on x_* and each of the training points in X.

$$K(X, x_*) = \text{exp}(-\sigma \|X - x_*\|^2)$$

β is the intercept of the SVM.
ANSWER TASK 3: Using the decision function, we can classify an email as spam accordingly.

$$class = \begin{cases} \text{nonspam}, & \text{if sign}(\hat{y}(x_*)) = - \\ \text{spam}, & \text{if sign}(\hat{y}(x_*)) = + \end{cases}$$

The result of the SVM predictions on the first 10 points in the spam dataset can be seen in table 1

Data Point	**Prediction (Class)**	**Decision Value ($\hat{y}(x_*)$)**
1	Nonspam	-1.0703
2	Spam	1.0003
3	Spam	0.9996
4	Nonspam	-0.9999
5	Nonspam	-0.9995
6	Spam	1.0001
7	Nonspam	-0.8586
8	Nonspam	-0.9997
9	Spam	0.9998
10	Nonspam	-1.0001

Table 1: SVM Predictions for the First 10 Data Points in the spam dataset

9 Lab 3 Task 3: Neural Networks

This assignment is to be solved using the neuralnet package.

Tasks

1. Train a neural network to learn the trigonometric sine function.

- Sample 500 points uniformly at random in the interval $[0, 10]$.
- Apply the sine function to each point. The resulting value pairs are the data points available to you.
- Use 25 of the 500 points for training and the rest for testing.
- Use one hidden layer with 10 hidden units. You do not need to apply early stopping.
- Plot the training and test data, and the predictions of the learned neural network on the test data.
- Comment on your results.

2. In question (1), you used the default logistic (a.k.a. sigmoid) activation function, i.e., act.fct = "logistic".
   - Repeat question (1) with the following custom activation functions:
     $$h_1(x) = x, \quad h_2(x) = \max\{0, x\}, \quad h_3(x) = \ln(1 + e^x)$$
     (a.k.a. linear, ReLU, and softplus).
   - See the help file of the neuralnet package to learn how to use custom activation functions.
   - Plot and comment on your results.
3. Sample 500 points uniformly at random in the interval $[0, 50]$ and apply the sine function to each point.
   - Use the neural network learned in question (1) to predict the sine function value for these new 500 points.
   - You should get mixed results. Plot and comment on your results.
4. In question (3), the predictions seem to converge to some value.
   - Explain why this happens. To answer this question, you may need to access the weights of the learned neural network.
   - You can do this by running nn or nn$weights, where nn is the learned neural network.
5. Sample 500 points uniformly at random in the interval $[0, 10]$ and apply the sine function to each point.
   - Use all these points as training points for learning a neural network that predicts x from $\sin(x)$, i.e., unlike before when the goal was to predict $\sin(x)$ from x.
   - Use the learned neural network to predict the training data.
   - You should get bad results. Plot and comment on your results.
   - **Help:** Some people get a convergence error in this question. This can be solved by stopping the training before reaching convergence by setting threshold = 0.1.

```
############################ Packages ############################

library(neuralnet)

############################ Part 1 ############################

n = 500
set.seed(1234567890)
```

```r
points = runif(n, min=0, max=10)
data <- data.frame(points, sin=sin(points))

# Split the dataset into 25 points for training and rest for testing
training_amount = 25
train_data = data[1:training_amount, ]
test_data = data[training_amount:n, ]

# Random initialization of the weights in the interval [-1, 1]
winit = runif(10, -1, 1)
nn = neuralnet(formula = sin ~ ., data = train_data, hidden = 10, startweights = winit)

# Plot of the training data (black), test data (blue), and predictions (red)
plot(train_data, cex=2, main = "Predictions of sin(x) from x")
points(test_data, col = "blue", cex=1)
points(test_data[, 1], predict(nn, test_data), col="red", cex=1)

legend("bottomright",
       legend = c("Training Data", "Test Data", "Predictions"),
       col = c("black", "blue", "red"),
       pch = 1,
       pt.cex = c(2, 1, 1))

# it can be seen that the model's predictions for the sine points using the test data are highly
↪   accurate. The predicted values closely overlap with the actual test data, indicating strong
↪   model performance.

########################### Part 2 ###########################

h1 = function(x) x # Linear
h2 = function(x) ifelse(x >= 0, x, 0) # ReLU
h3 = function(x) log(1 + exp(x)) # Softplus
nn1 = neuralnet(formula = sin ~ ., data = train_data, hidden = 10, act.fct = h1, startweights =
↪   winit)
nn2 = neuralnet(formula = sin ~ ., data = train_data, hidden = 10, act.fct = h2, startweights =
↪   winit)
nn3 = neuralnet(formula = sin ~ ., data = train_data, hidden = 10, act.fct = h3, startweights =
↪   winit)

plot(train_data, cex=2, main = "Predictions of sin(x) from x")
points(test_data, col = "blue", cex=1)
points(test_data[, 1], predict(nn1, test_data), col="red", cex=1)
points(test_data[, 1], predict(nn2, test_data), col="green", cex=1)
points(test_data[, 1], predict(nn3, test_data), col="orange", cex=1)

legend("bottomright",
       legend = c("Training Data", "Test Data", "NN1 (Linear)", "NN2 (ReLU)", "NN3
       ↪   (Softplus)"),
       col = c("black", "blue", "red", "green", "orange"),
       pch = 1,
       pt.cex = c(2, 1, 1, 1, 1))
```

```
# Figure shows that the Linear model performs poorly in predicting sin(x) from the data
↪   points, as a linear model cannot fit a sinusoidal curve. The ReLU activation function
↪   shows promising results for the initial data points but becomes worse after four points.
↪   This decline occurs because ReLU is a piecewise linear function, meaning it is linear in
↪   segments. Therefore, it struggles to capture the smooth, continuous oscillations of a
↪   sinusoidal function. On the other hand, the Softplus activation function allows the neural
↪   network to perfectly predict the test data. This is because Softplus is a smooth
↪   approximation of ReLU, making it well-suited for modeling non-linear functions.

########################## Part 3 ##########################

set.seed(1234567890)
points = runif(n, min=0, max=50)
data <- data.frame(points, sin=sin(points))

predictions = predict(nn, data)

plot(data, cex=2, ylim = c(min(predictions), max(predictions)), main = "Predictions of sin(x)
↪   from x")
points(data[, 1], predictions, col="red", cex=1)

legend("topright",
      legend = c("New Data", "Predictions"),
      col = c("black", "red"),
      pch = 1,
      pt.cex = c(2, 1))

# As shown in Figure the original NN model predicts the new data points perfectly until the
↪   after approximately x = 10 points. This is because the NN model is not trained for points
↪   larger than 10, leaving it unable to generalize the behavior of the sine function outside
↪   its training range.

########################## Part 4 ##########################

nn$weights

#[[1]]
#[[1]][[1]]
#          [,1]       [,2]       [,3]       [,4]      [,5]       [,6]      [,7]      [,8]      #[,9]      [,10]
#[1,]  3.3046893 -0.3271777  0.4043669 -0.7669631 11.955191 -0.2057986 -6.472200
↪   7.904868 #-0.5708964 0.04342472
#[2,] -0.8033506 -0.8324709 -0.1499125 -0.8256431 -1.802597  0.7993943  3.082136
↪   -2.320714  #0.1543628 0.76288667

#[[1]][[2]]
#            [,1]
# [1,]  0.7993476
# [2,] -4.9649102
# [3,] -4.8295057
# [4,] 22.1703831
# [5,] -5.5590648
# [6,] -4.3747945
# [7,]  0.3489759
# [8,] -0.7224382
# [9,]  1.8612110
#[10,] -9.4390597
#[11,]  0.4400295

#When examining the NN weights, especially in the second layer, we can observe that most
↪    of them are #negative. This dominance of negative weights will influence the networks
↪    output, especially for #outputs outside the training range. When the NN encounters data
↪    it hasnt been trained on, it relies #heavily on the these learned weights to make
↪    predictions. The effect of this causes the network's #output to converge to a negative
↪    value for such unseen data.

############################ Part 5 ############################

set.seed(1234567890)
points = runif(n, min=0, max=10)
data <- data.frame(points, sin=sin(points))

nn = neuralnet(formula = points ~ ., data = data, hidden = 10, startweights = winit, threshold
↪    = 0.1)

plot(x = data[, 2], y = data[, 1], cex=2, main = "Predictions of x from sin(x)", xlab = "sin", ylab
↪    = "points")
points(x = data[, 2], y = predict(nn, data), col = "red", cex = 1)

legend("bottomleft",
       legend = c("Training Data", "Predictions"),
       col = c("black", "red"),
       pch = 1,
       pt.cex = c(2, 1))

# The result of the NN trying to predict x from sin(x) can be seen in Figure \ref{fig:a3_t5}.
↪    This happens because its easier to predict sin(x) from x because the sine function
↪    provides a predictable output for each input x. The other way around is much harder, for
↪    the NN to learn the inverse of sine function, which is periodic, meaning that multiple
↪    values of x produce the same sin(x).
```

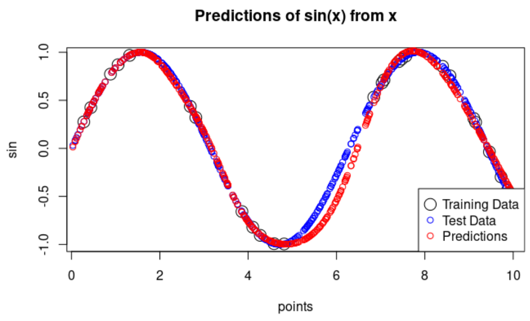

Figure 23: task 1

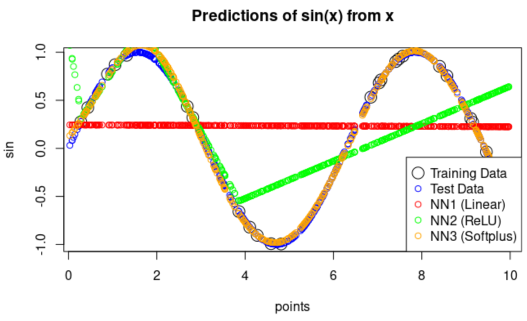

Figure 24: task 2

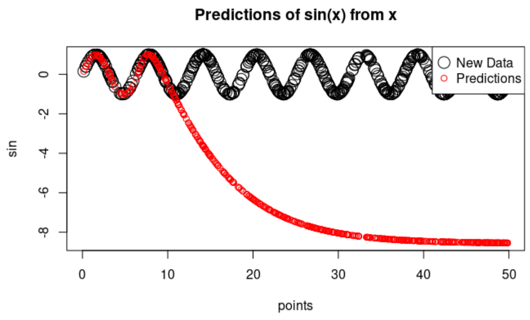

Figure 25: task 3

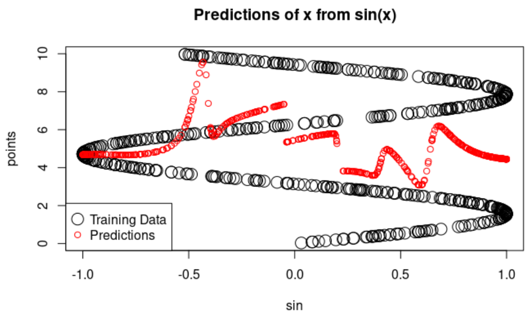

Figure 26: task 5

10 Exam 1

10.1 Assignment 1 (10p)

File Bikes.csv contains counts of public bicycles rented per hour in the Seoul Bike Sharing System, with corresponding weather data and holiday information.

1. Divide the data randomly into training and test data (70/30) and scale them appropriately. Compute a LASSO regression model by cross-validation in which Rented Bikes is the target variable and all remaining variables are features, and present a dependence of the cross-validation error on the penalty parameter. Which value of the penalty parameter is the optimal one? Interpret this model in terms of bias-variance tradeoff. Finally, report the equation showing how the predicted number of Rented bikes (scaled) depends on the features (scaled) in the estimated LASSO model corresponding to “lambda.1se” penalty value. **(4p)**
2. Consider *lambda.1se* LASSO model, use the training error MSE as an estimate of the target variance parameter and report the **95% prediction interval** for the first observation (first row) in the test data. Explain the model assumptions making these computations possible. **(2p)**
3. Consider the same partitioned and scaled data as in step 1. Assume now that Dew Point Temperature is related to the features Humidity and Visibility as a linear model without in-

tercept, and assume that the loss and the error functions are given by the following formula:

$$L(y, \hat{y}) = E(y, \hat{y}) = |y - \hat{y}|$$

Implement a code optimizing the cost function by the BFGS optimizer, and plot a dependence of the cost values and the test errors on the iteration number. Is early stopping needed? Report the optimal iteration number. Make 3 scatter plots of (Humidity, Visibility) where observations are colored by:

a) original target values,
b) predicted target values from the optimal model,
c) target values from the modelcorresponding to 5 iterations, and compare the plots with respect to the complexity of the model and quality of prediction. **(4p)**

Hint: to make scatter plots, you may use this kind of code from package ggplot2:

```
df = data.frame(x = your_variable1, y = your_variable2, color = your_variable3)
ggplot(df, aes(x = x, y = y, color = color)) + geom_point()
```

10.2 Assignment 1 Answers

```
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.6))
train=data[id,]
test=data[-id,]

library(caret)

#Assignment 1 Part 1
scaler=preProcess(train)
trainS=predict(scaler,train)
testS=predict(scaler, test)

covariates=trainS[,-1]
response=trainS[,1]

library(glmnet)

set.seed(12345)
model=cv.glmnet(as.matrix(covariates),
            response, alpha=1,family=&quot;gaussian&quot;)
model$lambda.min

coef(model, s="lambda.1se")

## 11 x 1 sparse Matrix of class "dgCMatrix"
##                             s1
## (Intercept)          -8.742707e-17
## Hour                 2.264222e-01
## Temperature          2.852890e-01
## Humidity            -9.808210e-02
## Wind.speed           .
## Visibility           .
## Dew.point.temperature .
## Solar.Radiation      .
## Rainfall            -1.702395e-02
## Snowfall            -6.135284e-02
## Holiday              1.086443e-01

#Assignment 1 Part 2
Ypred=predict(model, as.matrix(covariates), s="lambda.1se" )[,1]
MSE=mean((Ypred-response)^2)

testX=testS[1,-1]
Ytest=predict(model, as.matrix(testX), s="lambda.1se" )
CI=c(Ytest-1.96*sqrt(MSE), Ytest+1.96*sqrt(MSE))

print(CI)

## [1] -2.031268  1.240571

#Assignment 1 part 3

costF<-function(Y, Yfit){
  R=abs(Y-Yfit)
  return(mean(R))
}

x=as.matrix(trainS[,c(4,6)])
xt=as.matrix(testS[,c(4,6)])

Fs=list()
k=0
TestE=list()
Theta=list()

myCost= function(theta){
  f=costF(trainS$Dew.point.temperature, x%*%theta)
  .GlobalEnv$k= .GlobalEnv$k+1
  .GlobalEnv$Fs[[k]]=f
  .GlobalEnv$TestE[[k]]=costF(testS$Dew.point.temperature, xt%*%theta)
  .GlobalEnv$Theta[[k]]=theta
  return(f)
}

res=optim(rep(0,2), fn=myCost,  method="BFGS")


plot((as.numeric(Fs)), type="l", col="blue")
```

```
points((as.numeric(TestE)), type="l", col="red")

which.min(TestE)

## [1] 49

Pred1=xt%*%Theta[[which.min(TestE)]]
Pred2=xt%*%Theta[[5]]


df=data.frame(x=xt[,1], y=xt[,2], color=testS$Dew.point.temperature)
ggplot(df,aes(x=x,y=y,color=color))+geom_point()

df=data.frame(x=xt[,1], y=xt[,2], color=Pred1)
ggplot(df,aes(x=x,y=y,color=color))+geom_point()

df=data.frame(x=xt[,1], y=xt[,2], color=Pred2)
ggplot(df,aes(x=x,y=y,color=color))+geom_point()

#Note If the student makes these plots with training data or full data, no point reduction
↪    should be done since assignment does not specify which data to use in plots

#The optimal prediction model captures the trend and the original scale well, while the model
↪    with 4 iterations captures the trend but not the scale, all predictive values are near zero
↪    implying that the model is too simple for this data (underfitted).
```

10.3 Exam 1: Assignment 2 (10p)

10.3.1 EXERCISE 1 – 5 POINTS

In January 2023, the students were asked to implement the backpropagation algorithm for training a neural network for regression as it appears in the course textbook and slides. The solution is available to you in the file TDDO211January2023.R. Now, you are asked to incorporate dropout into this solution. Recall that dropout is a regularization technique whose detailed description you can find in the course textbook and slides.

Run your implementation with a dropout rate $r = 1$ equal to 0, 0.1, and 0.05, i.e., $r = 1, 0.99, 0.95$. Comment on the results.

10.3.2 EXERCISE 2 – 5 POINTS

You are asked to implement the perceptron algorithm. This algorithm for binary classification is a predecessor of modern neural networks.

Consider a binary classification problem with class labels $t \in \{-1, +1\}$. Then, the class label assigned to a point $\mathbf{z}$ is given by:

$$y(\mathbf{w}, \mathbf{z}) = \begin{cases} +1 & \text{if } \mathbf{w}^T\mathbf{z} \geq 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{z} < 0 \end{cases}$$

The values of **w** are iteratively determined with the help of the learning data $\{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$. Specifically, the i-th weight is updated in each iteration as follows:

$$w_i^{(t+1)} = w_i^{(t)} + \alpha \sum_{n=1}^{N} \Big(t_n - y(\mathbf{w}^{(t)}, \mathbf{z}_n)\Big)\, z_{n,i}$$

where α is the learning rate. Finally, you can assume that $\mathbf{z}_n$ and $\mathbf{w}$ are of dimension two, i.e., $\mathbf{z}_n = (x_{n,1}, x_{n,2})$ and $\mathbf{w} = (w_1, w_2)$.
You can stop the learning process after 100 iterations. You can use an alpha value of 0.0001. Plot the non-misclassification rate on the dataset below as a function of the number of iterations.

```
set.seed(1234)

x <- array(NA, dim = c(100, 2))

t <- array(NA, dim = c(100))

x[,1] <- runif(100,0,3)

x[,2] <- runif(100,0,9)

t <- ifelse(x[,2]<(x[,1]^2),-1,1)

plot(x[,1],x[,2],col=t+2)
```

Finally, explain when the perceptron algorithm works best and why it works in those cases.

10.4 Assignment 2 Answers

```
 # exercise 1 - 5 p
# 3 p for implementation
# 1 p if it runs correctly
# 1 p for answering the question correctly
# lines corresponding to dropout implementation are marked with ###, the rest of the code
↪    was provided in the exam

set.seed(1234)

# produce the training data in dat

x <- runif(500,-4,4)
y <- sin(x)
dat <- cbind(x,y)
plot(dat)

gamma <- 0.01
r <- 1 ### dropout rate 1-r
```

```
h <- function(z){

  # activation function (sigmoid)

  return(1/(1+exp(-z)))
}

hprime <- function(z){

  # derivative of the activation function (sigmoid)

  return(h(z) * (1 - h(z)))
}

yhat <- function(x){

  # prediction for point x

q0 <- x
z1 <- w1 %*% q0 + b1
q1 <- as.matrix(apply(z1,1,h), nrow = 2, ncol = 1)
z2 <- w2 %*% q1 + b2
return(z2)
}

yhat2 <- function(x){

  ### final prediction for point x (it involves r)

  q0 <- x
  z1 <- (r*w1) %*% q0 + b1
  q1 <- as.matrix(apply(z1,1,h), nrow = 2, ncol = 1)
  z2 <- (r*w2) %*% q1 + b2
  return(z2)
}

MSE <- function(){

  # mean squared error

  res <- NULL
  for(i in 1:nrow(dat)){
    res <- c(res,(dat[i,2] - yhat(dat[i,1])) ^ 2)
    }
  return(mean(res))
}

# initialize parameters

w1 <- matrix(runif(2,-.1,.1), nrow = 2, ncol = 1)
b1 <- matrix(runif(2,-.1,.1), nrow = 2, ncol = 1)
```

```r
w2 <- matrix(runif(2,-.1,.1), nrow = 1, ncol = 2)
b2 <- matrix(runif(1,-.1,.1), nrow = 1, ncol = 1)

res <- NULL
for(i in 1:100000){
  if(i %% 1000 == 0){
    res <- c(res,MSE())
  }

  # forward propagation

  j <- sample(1:nrow(dat),1)
  q0 <- dat[j,1]
  m0 <- sample(c(0,1),1,replace = TRUE, c(1-r,r)) ### drop input unit
  z1 <- w1 %*% (q0*m0) + b1
  q1 <- as.matrix(apply(z1,1,h), nrow = 2, ncol = 1)
  m1 <- sample(c(0,1),2,replace = TRUE, c(1-r,r)) ### drop hidden units
  z2 <- w2 %*% (q1*m1) + b2

  # backward propagation

  dz2 <- - 2 * (dat[j,2] - z2)
  dq1 <- t(w2) %*% dz2
  dz1 <- dq1 * hprime(z1)
  dw2 <- dz2 %*% t(q1)
  db2 <- dz2
  dw1 <- dz1 %*% t(q0)
  db1 <- dz1

  # parameter updating

  w2 <- w2 - (gamma * dw2 * m1) ### update non-dropped hidden units
  b2 <- b2 - gamma * db2
  w1 <- w1 - (gamma * dw1 * c(m0,m0) * m1) ### update non-dropped hidden units
  b1 <- b1 - gamma * db1
}
plot(res, type = "l")

plot(dat)
points(dat[,1],lapply(dat[,1],yhat2),col="red") ### apply final prediction

# Lower r value implies larger regularization and, thus, worse fit of the training data. The
↪    ultimate goal of dropout is
# to get a better fit of the test data (i.e., low generalization error). However, this is too simple
↪    an example to observe it.

# exercise 2 - 5 p
# 3 p for implementation
# 1 p if it runs correctly
# 1 p for answering the question correctly
```

```
set.seed(1234)

x <- array(NA, dim = c(100,2))
t <- array(NA, dim = c(100))
x[,1] <- runif(100,0,3)
x[,2] <- runif(100,0,9)
t <- ifelse(x[,2]<(x[,1])^2,-1,1)
plot(x[,1],x[,2],col=t+2)

w <- c(0,0)
alpha <- 0.0001

res <- NULL
for(i in 1:100){
  res <- c(res,sum(t != (2 * ((w %*% t(x[,1:2]))>0) - 1))) # note that t is both class label and
  ↪   transpose operation :-(

  w[1] <- w[1] + alpha * sum((t - w %*% t(x[,1:2])) * x[,1])
  w[2] <- w[2] + alpha * sum((t - w %*% t(x[,1:2])) * x[,2])
}

plot(x[,1],x[,2],col=(2 * ((w %*% t(x[,1:2]))>0) - 1)+2)
plot(res, type = "l")

# It works for linearly separable datasets. If the true label is larger than the prediction, then
↪   the weights get
# increased for positive input values and decreased for negative input values. The opposite
↪   when the true label is smaller.
```

11 Exam 2

11.1 Assignment 1 (10p)

File Rice.csv contains a total of 3810 rice grain's images taken for the two species (Cammeo and Osmancik), which were processed and feature inferences were made. 7 morphological features were obtained for each grain of rice which are the variables in the data set.

1. Divide the data randomly into training and test data (70/30). Assume that columns Area,...,Extent are denoted as $x_1, ..., x_p$. Perform basis function expansion with basis functions $\phi_1 = x_1, ..., \phi_p = x_p, \phi_{p+1} = x_1^2, ...\phi_{2p} = x_p^2$ and fit two logistic regression models: one with features $x_1, ..., x_p$ and another with $\phi_1, ..., \phi_{2p}$, both with target Class to the training data. Report the training and test misclassification errors for both models and report which model is better and why. Finally, report the estimated probabilistic model for the model with features $x_1, ..., x_p$. **(4p)**
2. Use cross-validation to fit a decision tree model to the training data with target Class and all other columns ($x_1, ..., x_p$) as features and report the optimal number of leaves and training and test misclassification errors for the optimal tree. Comment on the prediction quality of the optimal tree. After this, write a loop that grows decision trees with parameter mindev =

0.001, 0.002, ... 0.01 (without cross-validation) and estimates training and test misclassification errors for these trees. Plot the dependence of these errors on mindev, comment on the trends observed in the plot and comment how the trends are expected to look in theory and why. **(4p)**

3. Use the model estimated by the cross-validation in step 2 and assume that a user wants to prune this tree. If we use misclassification error as impurity measure, which leaves must be pruned first, 14 and 15, or 4 and 5? Report necessary mathematical calculations to support your answer. **(2p)**

11.2 Assigment 1: Answers

```
#Part 1

df=read.csv("Rice.csv", stringsAsFactors = T)
library(dplyr)

n=dim(df)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.7))
train=df[id,]
test=df[-id,]


train1=train%>%mutate_if(is.numeric, list(x2=function(x) x^2))
test1=test%>%mutate_if(is.numeric, list(x2=function(x) x^2))

m1=glm(Class~., data=train, family = binomial)

m2=glm(Class~., data=train1, family=binomial)

missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}



missclass(train$Class, predict(m1, newdata = train, type="response")>0.5)

# [1] 0.06861642

missclass(test$Class, predict(m1, newdata = test, type="response")>0.5)

# [1] 0.07524059

missclass(train$Class, predict(m2, newdata = train1, type="response")>0.5)

# [1] 0.06786652
```

```
missclass(test$Class, predict(m2, newdata = test1, type="response")>0.5)

# [1] 0.07611549

#We can see that after basis function expansion the training error gets smaller and test error
↪    gets larger which indicates overfitting. The model with original features is best one

m1

##
## Call:  glm(formula = Class ~ ., family = binomial, data = train)
##
## Coefficients:
##       (Intercept)               Area          Perimeter  Major_Axis_Length
##         -7.910612           0.006971           0.107500          -0.112395
## Minor_Axis_Length       Eccentricity        Convex_Area             Extent
##          0.553997          -3.747786          -0.011754           0.315431
##
## Degrees of Freedom: 2666 Total (i.e. Null);  2659 Residual
## Null Deviance:       3656
## Residual Deviance: 920.9     AIC: 936.9

levels(train$Class)

## [1] "Cammeo"   "Osmancik"

#
```

The probabilistic model is

$$P(\text{Class} = \text{Osmancik}) = \frac{1}{1+\text{exp}(z)}$$

where

$z = 7.910612 - 0.006971 \cdot \text{Area} - 0.107500 \cdot \text{Perimeter} + 0.112395 \cdot \text{MajorAxisLength} - 0.553997 \cdot \text{MinorAxisLength} + 3.7$

```
#Part 2
library(tree)
fit=tree(Class~., data=train)

set.seed(12345)
cv.res=cv.tree(fit)
finalTree0=prune.tree(fit, best=cv.res$size[which.min(cv.res$dev)])
Yfit1=predict(finalTree0, newdata=train,
            type="class")
Yfit2=predict(finalTree0, newdata=test,
             type="class")
```

```
cv.res$size[which.min(cv.res$dev)]

# [1] 5

missclass(train$Class,Yfit1)

# [1] 0.06861642

missclass(test$Class,Yfit2)

# [1] 0.07786527

#The tree has a good prediction quality since test error is only around 7 percent.

mind=seq(0.001, 0.01, 0.001)
IM=length(mind)
TrE=numeric(IM)
TeE=numeric(IM)

for (i in 1:IM){
  fit=tree(Class~., data=train, control=tree.control(nobs=nrow(train),mindev=mind[i]) )
  TrE[i]=missclass(predict(fit, newdata=train, type="class"), train$Class)
  TeE[i]=missclass(predict(fit, newdata=test, type="class"), test$Class)

}

plot(mind, TrE, col="blue", ylim=c(0.05,0.15))
points(mind, TeE, col="red")
```

```
#Part 3

print(finalTree0)

## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 2667 3656.0 Osmancik ( 0.437945 0.562055 )
##    2) Major_Axis_Length < 191.075 1488  657.3 Osmancik ( 0.057796 0.942204 )
##      4) Major_Axis_Length < 181.225 1116  141.6 Osmancik ( 0.011649 0.988351 ) *
##      5) Major_Axis_Length > 181.225 372  368.4 Osmancik ( 0.196237 0.803763 ) *
##    3) Major_Axis_Length > 191.075 1179  670.3 Cammeo ( 0.917727 0.082273 )
##      6) Perimeter < 467.639 198  258.4 Cammeo ( 0.641414 0.358586 ) *
##      7) Perimeter > 467.639 981  240.1 Cammeo ( 0.973496 0.026504 )
##       14) Major_Axis_Length < 205.866 409  188.2 Cammeo ( 0.938875 0.061125 ) *
##       15) Major_Axis_Length > 205.866 572   14.7 Cammeo ( 0.998252 0.001748 ) *

ImpurityDiff0=0.057796*1488-(0.011649*1116  +0.196237*372)
ImpurityDiff0

## [1] 1.421085e-14
```

```
ImpurityDiff1=0.026504*981  -(0.061125*409+0.001748*572 )
ImpurityDiff1

## [1] 0.000443

#4 and 5 need to be pruned first since provide smallest impurity decrease
```

11.3 Assignment 2 (10p)

You are asked to implement the backpropagation algorithm for training a neural network for regression as it appears in the course textbook and slides. To do so, fill in the code provided below. The actual network has one hidden layer with two units. x denotes inputs, y denotes targets, $\hat{y}$ denotes activation units, and z denotes hidden units. As the result of applying the activation function to z, $\sigma(z)$ denotes the squashed error, and J denotes the loss (mean squared error). The steps involved in this task are: (1) forward propagation, (2) backward propagation, (3) parameter update as marked in substeps 1-3B. Except the one indicated with a *, that is element-wise product. (3B). Notice the use of matrix transposition in some steps (3B) in R. Comment your code and add results. The exercise will be graded as follows: Forward propagation 2p, backward propagation 4p, parameter updating 1p, and results 3p.

Steps:

- **Forward propagation.**

$$q^{(0)} = x$$
$$z^{(1)} = W^{(1)}q^{(0)} + b^{(1)}$$
$$q^{(1)} = h(z^{(1)})$$
$$z^{(2)} = W^{(2)}q^{(1)} + b^{(2)}$$
$$J(\theta) = (y - z^{(2)})^2$$

- **Backward propagation.**

$$dz^{(2)} = -2(y - z^{(2)})$$
$$dq^{(1)} = W^{(2)T}dz^{(2)}$$
$$dz^{(1)} = dq^{(1)} \odot h'(z^{(1)})$$
$$dW^{(2)} = dz^{(2)}q^{(1)T}$$
$$db^{(2)} = dz^{(2)}$$
$$dW^{(1)} = dz^{(1)}q^{(0)T}$$
$$db^{(1)} = dz^{(1)}$$

- **Parameter updating.**

$$W^{(2)}_{t+1} = W^{(2)}_t - \gamma dW^{(2)}$$
$$b^{(2)}_{t+1} = b^{(2)}_t - \gamma db^{(2)}$$
$$W^{(1)}_{t+1} = W^{(1)}_t - \gamma dW^{(1)}$$
$$b^{(1)}_{t+1} = b^{(1)}_t - \gamma db^{(1)}$$

Code Template:

You are requested to use the template below. Note that the algorithm performs 100000 iterations. In each iteration, one randomly selected training point is used to update the parameters (i.e., this essentially corresponds to stochastic gradient descent with a mini-batch of size 1).

```r
# produce the training data in R
x = runif(500,-4,4)
y = sin(x)
dat = cbind(x,y)
plot(dat)

gamma = 0.01

h = function(z){
  # activation function (sigmoid)
  return(1/(1+exp(-z)))
}

hprime = function(z){
  # derivative of the activation function (sigmoid)
  return(h(z) * (1 - h(z)))
}

yhat <- function(x){
  # prediction for point x
}

MSE <- function(res){
  # mean squared error
}

# initialize parameters
res <- NULL
for(it in 1:100000){
  if(it %% 10000 == 0){
    res <- c(res, MSE())
  }

  # forward propagation
  j <- sample(1:nrow(dat),1)
  x0 <- dat[j,1]
  # backward propagation
  # parameter updating
}

plot(res, type = "l")
ploat(dat)
points(dat[,1],apply(dat[,1],1,yhat))
```

11.4 Assignment 2: Answers

```
    set.seed(1234)

# produce the training data in dat

x <- runif(500,-4,4)
y <- sin(x)
dat <- cbind(x,y)
plot(dat)

gamma <- 0.01

h <- function(z){

  # activation function (sigmoid)

  return(1/(1+exp(-z)))
}

hprime <- function(z){

  # derivative of the activation function (sigmoid)

  return(h(z) * (1 - h(z)))
}

yhat <- function(x){

  # prediction for point x

q0 <- x
z1 <- w1 %*% q0 + b1
q1 <- as.matrix(apply(z1,1,h), nrow = 2, ncol = 1)
z2 <- w2 %*% q1 + b2
return(z2)
}

MSE <- function(){
  # mean squared error

  res <- NULL
  for(i in 1:nrow(dat)){
    res <- c(res,(dat[i,2] - yhat(dat[i,1])) ^ 2)
    }
  return(mean(res))
}

# initialize parameters

w1 <- matrix(runif(2,-.1,.1), nrow = 2, ncol = 1)
b1 <- matrix(runif(2,-.1,.1), nrow = 2, ncol = 1)

w2 <- matrix(runif(2,-.1,.1), nrow = 1, ncol = 2)
b2 <- matrix(runif(1,-.1,.1), nrow = 1, ncol = 1)

res <- NULL
for(i in 1:100000){
  if(i %% 1000 == 0){
    res <- c(res,MSE())
  }

  # forward propagation

  j <- sample(1:nrow(dat),1)
  q0 <- dat[j,1]
  z1 <- w1 %*% q0 + b1
  q1 <- as.matrix(apply(z1,1,h), nrow = 2, ncol = 1)
  z2 <- w2 %*% q1 + b2

  # backward propagation

  dz2 <- - 2 * (dat[j,2] - z2)
  dq1 <- t(w2) %*% dz2
  dz1 <- dq1 * hprime(z1)
  dw2 <- dz2 %*% t(q1)
  db2 <- dz2
  dw1 <- dz1 %*% t(q0)
  db1 <- dz1

  # parameter updating

  w2 <- w2 - gamma * dw2
  b2 <- b2 - gamma * db2
  w1 <- w1 - gamma * dw1
  b1 <- b1 - gamma * db1
}
plot(res, type = "l")

plot(dat)
```

```
points(dat[,1],lapply(dat[,1],yhat),col="red")
```

12 Theory

12.1 Linear Model

The general formula for a linear model is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon$$

In matrix form, the same model can be expressed as:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

Where:

- y: Dependent variable (response).**X**: Design matrix of predictors, of size $n \times (p+1)$, where the first column is typically ones (for the intercept).
- β_0: Intercept.
- $\beta_1, \beta_2, \ldots, \beta_p$: Coefficients for the independent variables $x_1, x_2, \ldots, x_p$. $\boldsymbol{\beta}$: Coefficient vector, including the intercept, of size $(p+1) \times 1$.
- ϵ: Error term, capturing the variation not explained by the model. $\boldsymbol{\epsilon}$: Vector of error terms, of size $n \times 1$.
- **y**: Vector of responses, of size $n \times 1$.

12.2 Logistic Regression

The probability of the positive class is given by:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)}}$$

In terms of the logit function:

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

In matrix form:

$$p = \frac{1}{1 + e^{-\mathbf{X}\boldsymbol{\beta}}}$$

12.3 Bias Variance tradeoff

1. Bias:
   - Bias refers to the error introduced by approximating a real-world problem, which may be highly complex, by a simplified model.
   - High bias implies that the model is too simplistic and may not capture the underlying patterns in the data.
   - This can lead to systematic errors, and the model may consistently underperform.
2. Variance:
   - Variance refers to the model's sensitivity to small fluctuations or noise in the training data.
   - High variance implies that the model is too complex and has learned to capture the noise in the training data rather than the underlying patterns.
   - This can lead to overfitting, where the model performs well on the training data but poorly on new, unseen data.

The tradeoff arises because increasing the complexity of a model typically reduces bias but increases variance, and vice versa. The goal is to find the right balance that minimizes the overall prediction error on new, unseen data. Bayes variance: the more leaves, the higher variance but smaller bias. If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

12.4 Precision and recall

The formulas for Precision and Recall are as follows:

Precision measures the proportion of positive predictions that were actually correct. It focuses on the accuracy of the positive predictions made by the model.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall measures the proportion of actual positive instances that were correctly identified by the model. It focuses on the completeness of the positive predictions.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

12.5 F1 score

Good measure of performance if the dataset is imbalanced. Measures how good the model is at guessing "positive". The formula for the F1 score is given by:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

13 Old exams

The file lakesurvey.csv contains chemical measurements of different lakes in Sweden as well as their pH level.

Question 1 (3p)

1. Perform principal component analysis (PCA) excluding the pH variable on the scaled original data.
2. How much variation is explained by the first two principal components?
3. Which features contribute mostly to the first principal component?
4. By assuming that we keep only the first two principal components, report an equation showing how the unscaled Cond variable can be approximated from the first two principal components.

Question 2 (5p)

1. Divide the original data into training, validation, and test sets (40/30/30).
2. Consider also a dataset with the three variables: the first two principal components and the pH variable. Split this dataset into training, validation, and test sets by using the same partitioning indices as in the original dataset.
3. Scale the datasets appropriately.
4. Estimate the following models with pH as the target variable and the remaining variables as features:
   (a) K-nearest neighbor (KNN) models using the original data partitioning with $K = 1, 10, 50$.
   (b) K-nearest neighbor (KNN) models using the principal component data partitioning with $K = 1, 10, 50$.
5. Report training, validation, and test MSE for these 6 models.
6. Answer:
   - Which K leads to the best model from the original data?
   - Which K leads to the best model from the principal component dataset?
   - What kind of data (original or principal component) results in the best prediction?
   - Why might one need to check the test error of the best model as well?
   - Why does $K = 1$ result in zero training MSE?
   - What target distribution do we implicitly assume by using MSE as a cost function?

Question 3 (2p)

1. Use the original dataset and the principal component dataset from step 2. Scale them, and compute the ratio between:
   - the Euclidean distance to the nearest observation from observation number 1, and
   - the Euclidean distance to the furthest observation from observation number 1.

   (Exclude the pH variable in the distance computations.)
2. Compare these ratios and comment on what kind of machine learning phenomenon this comparison illustrates.

```
setwd("~/Repos/machine-learning-tdde01/solutions/2025-01-17")

### library ###
library(dplyr)
library(caret)
library(kknn)

### Task 1 ###
data_frame = read.csv("lakesurvey.csv") #column , seperators
df_no_ph = select(data_frame, !pH & !LakeID)
scaler = preProcess(df_no_ph)
df_no_ph_s = predict(scaler, df_no_ph)

pca_res = princomp(na.omit(df_no_ph_s)) # na.omit(data) remove NA values
summary(pca_res)

# How much variation is explained by the first two principal components?
# Variance for component 1 and 2 is
# Proportion of Variance comp 1: 0.4306978 comp2: 0.1978562

loadings(pca_res) # Inspect loadings

features_pca_1 = pca_res$loadings[, 1]
abs(features_pca_1)

# Which features contribute to the first principal component mostly?
# The ones with absolute larger number contribute more to the first component.
# > pca_res$loadings[, 1]

# By assuming that we keep only first two principal
# components, report an equation showing how the unscaled Cond variable can be
# approximated from the first two principal components

df_no_ph_cc <- na.omit(df_no_ph) # 1) Remove rows with missing values
# Calculate the original mean for the 'Cond' variable before PCA
mean_cond_original <- mean(df_no_ph_cc$Cond)

pca_raw <- princomp(df_no_ph_cc, cor = FALSE)  # 2) Run PCA on raw data (no scaling,
↪    but centered by default)
# summary(pca_raw) # 3) Summary of variance explained
# loadings(pca_raw) # 4) Inspect loadings (in original units)
scores_raw <- pca_raw$scores[, 1:2]  # 5) Get PC scores for first two PCs (these are for
↪   centered data)

# 6) Get the loadings for the 'Cond' variable for the first two PCs
l_cond <- pca_raw$loadings["Cond", 1:2]

# 7) Approximate Cond from first two PCs, this result is for the *centered* Cond variable
Cond_hat_centered <- scores_raw %*% l_cond

# 8) Add back the original mean of 'Cond' to get the unscaled and uncentered approximation
Cond_hat_final <- Cond_hat_centered + mean_cond_original

# 9) View first 10 comparisons
head(cbind(Cond_true = df_no_ph_cc$Cond,
         Cond_hat_approx = Cond_hat_final), 10)

# The equation showing how the unscaled Cond variable can be approximated:
# Cond_approximated ≈ (lCond1 * PC1) + (lCond2 * PC2) + Mean(Cond_original)
# Where PC1 and PC2 are the scores (values from the principal co


### Task 2 ###

two_pca = pca_res$scores[, 1:2]
data_combined = cbind(two_pca, select(data_frame, pH)) # Dataset 3 variables

n = dim(data_frame)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.4))
data_train = data_frame[id, ]
combined_train = data_combined[id, ]

id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n * 0.3))
data_validation = data_frame[id2, ]
combined_validation = data_combined[id2, ]

id3 = setdiff(id1, id2)
data_test = data_frame[id3, ]
combined_test = data_combined[id3, ]

# Scale original dataset
scaler_original = preProcess(data_train)
data_train = predict(scaler_original, data_train)
data_validation = predict(scaler_original, data_validation)
data_test = predict(scaler_original, data_test)

# Scale the 3 variable combined dataset
scaler_combined = preProcess(combined_train)
combined_train = predict(scaler_combined, combined_train)
```

```r
combined_validation = predict(scaler_combined, combined_validation)
combined_test = predict(scaler_combined, combined_test)

kknn_mse = function(train_data, test_data, k_val) {
  kknn_model = kknn(formula = pH ~ .,
                    train = train_data, test = test_data,
                    k = k_val, kernel = "rectangular")
  predicts_kknn = predict(kknn_model)
  mse = mean((test_data$pH - predicts_kknn)^2)
  print(mse)
  return(mse)
}

k_values = c(1, 10, 50)
mse_original = data.frame("train_original" = 0, "validation_original" = 0, "test_original" = 0)
mse_pca = data.frame("train_pca" = 0, "validation_pca" = 0, "test_pca" = 0)
index = c(0)

for(k in k_values) {
  index = index + 1
  mse_original[index, 1] <- kknn_mse(data_train, data_train, k)
  mse_original[index, 2] <- kknn_mse(data_train, data_validation, k)
  mse_original[index, 3] <- kknn_mse(data_train, data_test, k)
  mse_pca[index, 1] <- kknn_mse(combined_train, combined_train, k)
  mse_pca[index, 2] <- kknn_mse(combined_train, combined_validation, k)
  mse_pca[index, 3] <- kknn_mse(combined_train, combined_test, k)
}
mse_original
mse_pca
# > mse_original
# train_original validation_original test_original
# 1 0.0000000 0.4367774 0.4813599
# 2 0.2479928 0.3214056 0.3370375
# 3 0.3852931 0.4285126 0.4746325
# > mse_pca
# train_pca validation_pca test_pca
# 1 0.0000000 0.8685921 0.8221548
# 2 0.4040912 0.4754348 0.5049334
# 3 0.4893246 0.5297236 0.5755813

# Answer:
# 1. Which K leads to the best model from the original data?
# k = 10 leads to the best model because the mean squared error is the lowest

# 2. Which K leads to the best model from the PCA dataset?
# K = 10 leads also to the best model from the PCA dataset with the lowest validation error.

# 3. Which dataset gives the best prediction?
# The original dataset

# 4. Why might one need to check the test error of the best model?
# We select the model based on training, validation error. But to estimate its generalization
↪   ability on unseen
```

```
# data we check the test error. The test set gives an unbiased estimate how the model will
↪   perform in practise.

# 5. Why does K=1 result in zero training MSE?
# When K = 1 each point in the training set is its own nearest neighbor, so the prediction is
↪   exactly the same as the true value.
# This makes the training MSE zero but it usually leads to overfitting and poor generalization.

# 6. What target distribution do we implicitly assume by using MSE?
# By using MSE as a cost function, we implicitly assume that the target variable is normally
↪   distributed
# around the regression function with constant variance. MSE corresponds to the maximum
↪   likelihood estimator
# under gaussian noise assumptions.


############### Task 3 ###############
# Original data without pH
original_features = select(data_frame, -pH, -LakeID)
original_scaled = predict(preProcess(original_features), original_features)

# PCA data (already only two PCs + pH); exclude pH
pca_features = select(data_combined, -pH)
pca_scaled = predict(preProcess(pca_features), pca_features)

euclidean_ratio <- function(data) {
  d <- as.matrix(dist(data), method = "euclidean")
  d1 <- d[1, -1] # selects all distances from observation 1 to every other observation (exclude
  ↪   itself).
  min(d1) / max(d1) # Ratiop nearest / farthest distance from observation 1
}

ratio_original = euclidean_ratio(original_scaled)
ratio_pca = euclidean_ratio(pca_scaled)
ratio_original
ratio_pca
# Comment:
# Higher ratio (closer to 1) in original high-dimensional space shows distance concentration.
# Lower ratio after PCA (2D) shows better separation of near vs far neighbors.
```

1. **KERNEL MODELS – 6 Points**
   - Kernel models can be used for density estimation:

$$p(x^*) = \frac{1}{n}\sum_{i=1}^{n} k\left(\frac{x^* - x_i}{h}\right)$$

     with $k()$ being a Gaussian density function.
   - **(2 p)** Generate learning data (1000 samples per class) in R and use 800 samples per class to estimate class conditional densities with the kernel model.

- **(3 p)** Compute posterior class probabilities via Bayes theorem:

$$p(class = 1|x^*) = \frac{p(x^*|class = 1)\, p(class = 1)}{p(x^*|class = 1)\, p(class = 1) + p(x^*|class = 2)\, p(class = 2)}$$

  Use 200 new samples (100 per class) to compute classification rate and select kernel width $h \in \{0.1, 0.2, \dots, 5\}$.
- **(1 p)** Estimate generalization error using the remaining 200 samples.
- **Summary:**
  - Use 1600 samples for kernel density estimation.
  - Convert densities to a probabilistic classifier using Bayes theorem.
  - Use 200 samples for validation (kernel width selection).
  - Use 200 samples for generalization error estimation.

2. **NEURAL NETWORKS – 4 Points**
   - Sample 50 points uniformly in $[0, 10]$ and compute their sine values. Train a neural network with a single hidden layer of 10 units.
   - **(3 p)** Estimate generalization MSE using 2-fold cross-validation:
     a) Divide data into folds D_1 and D_2.
     b) Train on D_1, test on D_2.
     c) Train on D_2, test on D_1.
     d) Report average test error.

     Initialize weights randomly in $[-1, 1]$ and stop training when partial derivatives $<$ 0.001.
   - **(1 p)** Decide which NN to return to the user: from D_1, D_2, either, or none.
   - **Notes:** Use threshold in neuralnet, predict() for predictions, and default arguments otherwise. Comment your code.

```
##########################
# Kernel methods - TDDE01 - 6p
# This script:
# 1. Generates 1D synthetic data for two classes (mixture distributions)
# 2. Implements simple (teacher-style) kernel density estimates using first 800 samples
↪   (train)
# 3. Validates bandwidth h on next 100 samples (validation)
# 4. Retrains using 900 samples (train+validation) and estimates test accuracy on last 100
# 5. Performs a simple 2-fold NN cross-validation, then trains final NN
##########################

set.seed(123456789)

N_class1 <- 1000
N_class2 <- 1000

# Generate class 1:
# For each point: with prob 0.3 draw from N(15,3^2), else from N(4,2^2)
data_class1 <- NULL
for(i in 1:N_class1){
  a <- rbinom(n = 1, size = 1, prob = 0.3)              # mixture component indicator
  b <- rnorm(n = 1, mean = 15, sd = 3) * a +
       (1-a) * rnorm(n = 1, mean = 4, sd = 2)
  data_class1 <- c(data_class1,b)
}

# Generate class 2:
# With prob 0.4 draw from N(10,5^2), else from N(15,2^2)
data_class2 <- NULL
for(i in 1:N_class2){
  a <- rbinom(n = 1, size = 1, prob = 0.4)
  b <- rnorm(n = 1, mean = 10, sd = 5) * a +
       (1-a) * rnorm(n = 1, mean = 15, sd = 2)
  data_class2 <- c(data_class2,b)
}

# ----- Kernel density estimates (training on first 800 points) -----
# NOTE: Standard KDE formula is (1/(n*h)) * sum K((t - xi)/h).
# The teacher formula given in assignment omits the 1/h factor, so we follow that
↪    specification.

conditional_class1 <- function(t, h){
  d <- 0
  for(i in 1:800)
    d <- d + dnorm((t - data_class1[i]) / h)   # accumulate kernel contributions
  return (d / 800)                             # average (teacher version)
}

conditional_class2 <- function(t, h){
  d <- 0
  for(i in 1:800)
    d <- d + dnorm((t - data_class2[i]) / h)
  return (d / 800)
}

# ----- Posterior P(Class=1 | t) with equal implicit priors (800/1600) -----
prob_class1 <- function(t, h){
  prob_class1 <- conditional_class1(t, h) * 800 / 1600
  prob_class2 <- conditional_class2(t, h) * 800 / 1600
  return (prob_class1 / (prob_class1 + prob_class2))
}

# ----- Bandwidth selection via validation (points 801:900 in each class) -----
foo <- NULL
for(h in seq(0.1, 5, 0.1)){
  # Classification rule: predict class 1 if posterior > 0.5
  acc <- ( sum(prob_class1(data_class1[801:900], h) > 0.5) +
           sum(prob_class1(data_class2[801:900], h) < 0.5) ) / 200
  foo <- c(foo, acc)
}
plot(seq(0.1, 5, 0.1), foo)                     # validation accuracy vs h
```

```
max(foo)                                    # best validation accuracy
which(foo == max(foo)) * 0.1                    # corresponding h

# ----- Retrain densities using 900 (train+validation) points, test on last 100 -----

conditional_class1 <- function(t, h){
  d <- 0
  for(i in 1:900)
    d <- d + dnorm((t - data_class1[i]) / h)
  return (d / 900)
}

conditional_class2 <- function(t, h){
  d <- 0
  for(i in 1:900)
    d <- d + dnorm((t - data_class2[i]) / h)
  return (d / 900)
}

prob_class1 <- function(t, h){
  # Priors now 900/1800 each (still equal)
  prob_class1 <- conditional_class1(t, h) * 900 / 1800
  prob_class2 <- conditional_class2(t, h) * 900 / 1800
  return (prob_class1 / (prob_class1 + prob_class2))
}

h <- which(foo == max(foo)) * 0.1              # selected bandwidth
# Test accuracy on hold-out samples 901:1000 (100 per class)
(sum(prob_class1(data_class1[901:1000], h) > 0.5) +
 sum(prob_class1(data_class2[901:1000], h) < 0.5)) / 200

##########################
# Neural networks - TDDE01 - 4p
##########################

library(neuralnet)
set.seed(1234567890)

# Create regression data: y = sin(x) sampled at 50 random x values
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin = sin(Var))
tr1 <- trva[1:25,]   # fold 1
tr2 <- trva[26:50,]  # fold 2

# ----- 2-fold cross-validation for NN generalization error -----
# Each network: 1 input -> 10 hidden neurons -> 1 output
# Weight init length 31 = (1+1)*10 (input+ bias to hidden) + (10+1)*1 (hidden + bias to output)

winit <- runif(31, -1, 1)
nn1 <- neuralnet(formula = Sin ~ Var, data = tr1, hidden = 10,
                 startweights = winit, threshold = 0.001, lifesign = "full")
```

```
winit <- runif(31, -1, 1)
nn2 <- neuralnet(formula = Sin ~ Var, data = tr2, hidden = 10,
            startweights = winit, threshold = 0.001, lifesign = "full")

# Cross-predictions (each net evaluated on the other fold)
aux1 <- predict(nn1, tr2)
aux2 <- predict(nn2, tr1)

# Symmetric CV error (sum of MSE * 1/2 per fold)
sum((tr2[,2] - aux1)^2) / 2 + sum((tr1[,2] - aux2)^2) / 2

# ----- Final network trained on all data for deployment -----
winit <- runif(31, -1, 1)
nn1 <- neuralnet(formula = Sin ~ Var, data = rbind(tr1, tr2), hidden = 10,
            startweights = winit, threshold = 0.001, lifesign
```

Machine Learning Keywords by Source

From "Lecture 1a block 1.pdf"

Machine Learning Basics: Machine learning statistics prediction computational complexity data mining knowledge discovery data science artificial intelligence. Core Concepts: Data features targets mathematical model learning algorithm prediction maximum likelihood Bayesian estimation. Course Overview: Classification regression dimensionality reduction model selection kernel methods SVM neural networks mixture models ensemble methods. Learning Paradigms: Supervised learning unsupervised learning semi-supervised learning active learning reinforcement learning transfer learning. Key Models: Logistic regression K-nearest neighbor hyperparameter K decision boundaries. Evaluation and Challenges: Overfitting model selection holdout method training data test data validation data error functions MSE misclassification rate cross-entropy parametric models nonparametric models curse of dimensionality smoothness assumption.

From "Lecture 1c block 1.pdf"

R Environment: R programming language RStudio packages console workspace current directory help functions. Basic Operations: Case-sensitive comment variable assignment vectors matrices factors lists data frames. Matrix Operations: Transpose inverse solve indexing replication. Data Handling: Read CSV read Excel data frame to matrix numeric to factor. Control Flow and Functions: Loops if-else random number generation set.seed writing functions. Graphics: Plot histogram scatter plot graphical parameters color labels title.

From "Lecture 1d block 1.pdf"

Regression Models: Linear regression simple linear regression multiple linear regression logistic regression intercept regression coefficient. Model Learning (Linear): Ordinary

least squares cost minimization squared loss maximum likelihood hat matrix. Linear Regression in R: lm() summary() predict() coefficients(). Feature Engineering: Data scaling caret package feature types interval variables dummy coding basis function expansion polynomial regression. Regularization (General): Regularization overfitting penalize complexity. Specific Regularization Models: Ridge regression L2 regularization LASSO L1 regularization sparse solutions. Classification Concepts: Classification problem decision boundary deterministic classifiers probabilistic classifiers. Logistic Regression (Details): Sigmoid function logit link likelihood maximization Newton's method Steepest descent softmax linear discriminant analysis. Optimization: optim() BFGS.

From "Lecture 2a block 1.pdf"

Model Selection Principles: Model selection model evaluation estimator overfitting loss function error function data generating process expected risk. Validation Methods: Hold-out method training set validation set test set empirical risk cross-validation K-fold cross-validation leave-one-out. Performance Metrics: Generalization gap bias-variance tradeoff bias variance. Classification Evaluation: Confusion matrix accuracy true positive rate sensitivity recall false positive rate specificity precision ROC curves AUC. Imbalanced Data: Imbalanced classes F1 score FB score precision-recall curve.

From "Lecture 2b block 1.pdf"

Decision Tree Fundamentals: Decision trees hypercubes regression trees classification trees terminal nodes labels splitting rules. Learning Process: Greedy algorithm CART optimal tree postpruning weakest link pruning. Impurity Measures (Classification): Impurity measure misclassification rate Gini index cross-entropy deviance. Regression Tree Specifics: Mean squared error sum of squared residuals. Tree Characteristics: Tree size overfitting underfitting high variance lack of smoothness. Tree Strengths: Interpretable variable selection robust to outliers large datasets. Decision Trees in R: tree package rpart cv.tree() prune.tree().

From "Lecture 2c block 1.pdf"

Dimensionality Reduction Core: Dimensionality reduction latent variables feature reduction representation learning. Principal Component Analysis (PCA): PCA principal components maximizes variance eigenvalue decomposition eigenvectors eigenvalues scores loadings linear compression data scaling. PCA in R: prcomp() biplot() screeplot() trace plots. Non-linear/Probabilistic Approaches: Autoencoders nonlinear PCA probabilistic PCA independent component analysis ICA non-Gaussian distribution.

From "Lecture 2d block 1.pdf"

Parameter Estimation: Parametric functions cost minimization statistical noise. Loss Function Selection: Minus log-likelihood squared error absolute loss Huber loss E-insensitive loss cross-entropy exponential loss hinge loss. Multiclass Strategies: One versus one one versus rest majority voting. Regularization Types: Explicit regularization implicit

regularization early stopping L1 regularization L2 regularization. Ridge and LASSO (revisited): Ridge regression LASSO sparse solutions glmnet package. Optimization Algorithms: Numerical optimization gradient descent learning rate Newton's method Hessian quasi-Newton methods BFGS stochastic gradient descent mini-batches epoch. Hyperparameter Optimization: Grid search Bayesian optimization.

From "Lecture1b block 1.pdf"

Foundational Probability: Probability experiment outcomes sample space event probability function. Probability Rules: Set operations independence conditional probability sum rule product rule. Bayesian Inference: Bayes theorem prior probability likelihood posterior probability. Random Variables and Distributions: Random variables continuous random variables discrete random variables probability mass function probability density function cumulative distribution function. Statistical Measures: Expected value mean value variance. Common Distributions: Bernoulli distribution binomial distribution Poisson distribution Normal distribution Gaussian distribution multivariate distributions correlation. Model Fitting: Frequentist approach maximum likelihood principle log-likelihood Bayesian approach subjective probabilities conjugacy marginal likelihood. Uncertainty Quantification: Confidence interval credible interval prediction interval.

From "Lecture3aBlock12021.pdf"

Kernel Methods Overview: Kernel methods nonparametric kernel methods parametric kernel methods. Classification Approaches: Histogram classification moving window classification kernel classification majority vote. Kernel Functions: Kernel function smoothing factor width Gaussian kernel Cauchy kernel Epanechnikov kernel. Regression Approaches: Histogram regression moving window regression kernel regression kernel-weighted average. The Kernel Trick: Kernel trick feature space mapping $\phi(\cdot)$ inner products symmetric and positive semi-definite matrix. Kernel Ridge Regression: Kernel ridge regression ridge regression L2 regularization.

From "Lecture3bBlock12021.pdf"

Support Vector Regression (SVR): Support vector regression ε-insensitive loss function primal formulation dual formulation support vectors computation reduction ε hyperparameter representer theorem. Support Vector Classification (SVC): Support vector classification hinge loss function margin decision boundary. General SVM Concepts: Kernel trick feature space optimization problem numerical solution.

From "Lecture3cBlock12021.pdf"

Neural Networks Introduction: Neural networks linear regression model generalized linear regression model activation function hidden units two-layer NN multi-layer NN. NN Capabilities: Uniform approximation continuous function XOR problem M-class classification softmax. Backpropagation Algorithm: Backpropagation algorithm parameter optimization squared error loss cross-entropy loss gradient descent. Backpropagation Challenges

and Solutions: Parameter space multimodal initial parameter values stochastic gradient descent mini-batches forward propagation backward propagation gradient computation parameter updating.

From "Lecture3dBlock12021.pdf"

Convolutional Neural Networks (CNNs): Convolutional neural networks CNNs grid-like structure images spatial patterns dense layers sparse interactions parameter sharing invariance to translations stride pooling multiple channels. **NN Regularization: Dropout overfitting variance reduction bias L1 regularization L2 regularization early stopping sparse representations bagging**. **NN Strengths: Universal approximation theorem universal classification theorem arbitrary accuracy**. **NN Weaknesses: Hidden units training time local minimum No free lunch theorem**.