Assignment 3

2023-04-14

QA1. What is the difference between SVM with hard margin and soft margin?

Ans. Support Vector Machines is a machine learning model that helps to find the optimal decision boundary between two or more classes. The main difference between SVM with a hard margin and a soft margin is the flexibility of the decision boundary. In SVM with a hard margin, the algorithm seeks to find a decision boundary that perfectly separates the two classes without any misclassifications. However, this approach may only be feasible for some datasets. In such cases, SVM with a soft margin is used as it allows for misclassifications and a certain level of error to find a decision boundary that can still achieve a high level of accuracy while being more flexible. It is achieved by introducing a trade-off between maximizing the margin and minimizing the number of observed training errors. In other words, with SVM hard margin, the margin must be satisfied without exceptions. In contrast, with SVM soft margin, a small amount of error can achieve a more flexible decision boundary that can handle complex or overlapping datasets.

QA2. What is the role of the cost parameter, C, in SVM (with soft margin) classifiers?

Ans. Support Vector Machines is a popular machine learning algorithm used for classification problems. In SVM classification, the role of the C parameter is to control the trade-off between achieving a low training error and finding a decision boundary that generalizes well to new data. The C parameter determines the penalty for misclassified points in the training data. When the value of C is small, the SVM classifier allows more misclassifications in the training data and tends to generate a decision boundary that generalizes well to new data. On the other hand, when the value of C is large, the SVM classifier penalizes misclassifications heavily and produces decision boundaries that fit more tightly to the training data. Thus, selecting a suitable value for the C parameter is essential for obtaining optimal classification performance.

QA3. Will the following perceptron be activated (2.8 is the activation threshold) (10% of total points)

Ans. (0.1) *(0.8) = 0.08, 11.1 * (-0.2) = -2.22. 0.08 + - 2.22 = - 2.14. The result value is -2.14. Therefore, the perceptron will not be activated as the activation threshold is 2.8.

QA4. What is the role of alpha, the learning rate in the delta rule?

Ans. The delta rule is a commonly used algorithm in machine learning that updates the weights of neural network models to improve their accuracy. In the delta rule, alpha, the learning rate parameter, determines how much the weights are adjusted during each training iteration. The role of alpha, the learning rate in the delta rule, controls the step size taken during weight updates. If the alpha is set too high, the weight updates will be larger and may cause unstable or erratic behavior in the model. On the other hand, if alpha is set too low, the weight updates will be smaller and may take longer to converge on a solution.

```
library(ISLR)
library(glmnet)

## Loading required package: Matrix

## Loaded glmnet 4.1-7

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(dplyr)

##
## Attaching package: 'dplyr'

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

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

library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##     alpha

library(neuralnet)

##
## Attaching package: 'neuralnet'

## The following object is masked from 'package:dplyr':
##
##     compute
```

#Filtering variables for current project:

```
carseats <- Carseats %>% select("Sales", "Price",
"Advertising","Population","Age","Income","Education")
```

#QB1.Build a linear SVM regression model to predict Sales based on all other attributes

```
set.seed(123)
model_1<- train(Sales~.,data=carseats, method= "svmLinear")
print(model_1)

## Support Vector Machines with Linear Kernel
##
## 400 samples
##   6 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 400, 400, 400, 400, 400, 400, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   2.312705  0.3462025  1.858189
##
## Tuning parameter 'C' was held constant at a value of 1

#The values of C is "1" and R-squared values is 0.3462025.
```

#QB2. Customize the search grid by checking the model's performance for C parameter of 0.1,.5,1 and 10 using 2 repeats of 5-fold cross validation.

```
set.seed(123)
model_2 <- trainControl(method = "repeatedcv", number = 5, repeats = 2)
model_3 <- expand.grid(C = c(0.1, 0.5, 1, 10))

model_4<- train(Sales~., data = carseats, method = "svmLinear", trControl=
model_2, preProcess = c("center", "scale"), tuneGrid = model_3, tuneLength =
10)
model_4

## Support Vector Machines with Linear Kernel
##
## 400 samples
##   6 predictor
##
## Pre-processing: centered (6), scaled (6)
## Resampling: Cross-Validated (5 fold, repeated 2 times)
## Summary of sample sizes: 320, 321, 319, 320, 320, 319, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
##    0.1  2.299265  0.3462103  1.840099
##    0.5  2.298622  0.3466721  1.840422
##    1.0  2.300096  0.3457092  1.841191
##   10.0  2.298898  0.3464189  1.840055
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was C = 0.5.

#The best C for combination of 0.1, 0.5, 1, 10 is 0.5, R-squared being
0.3466721.
```

#QB3.Train a neural network model to predict Sales based on all other attributes

```
set.seed(123)
model_6<- train(Sales~., data = carseats, method = "nnet", preProcess =
c("center", "scale"),linout = TRUE, trace = FALSE)
model_6

## Neural Network
##
## 400 samples
##   6 predictor
##
## Pre-processing: centered (6), scaled (6)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 400, 400, 400, 400, 400, 400, ...
## Resampling results across tuning parameters:
##
##   size  decay  RMSE      Rsquared   MAE
##   1     0e+00  2.373011  0.3276504  1.879204
##   1     1e-04  2.352801  0.3202918  1.875685
##   1     1e-01  2.309111  0.3448201  1.844761
##   3     0e+00  3.042648  0.2319888  2.087725
##   3     1e-04  2.541338  0.2505847  2.033343
##   3     1e-01  2.483982  0.2701456  1.996496
##   5     0e+00  2.976992  0.1931621  2.220199
##   5     1e-04  2.651413  0.2222630  2.128007
##   5     1e-01  2.653569  0.2161488  2.140991
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 1 and decay = 0.1.
```

#QB4.What will be the estimated Sales for this record using the above neuralnet model?

```
new_carseats<- data.frame("Price" = 6.54, "Population" = 124, "Advertising" =
0, "Age" = 76, "Income" = 110, "Education" = 10)


model_7<- predict(model_6, new_carseats)
model_7

##        1
## 11.46014

#The estimated Sales for current data frame, using nnet model would be
11.46014.
```