Lab2_Classification

Lab 2

Method1: Support vector machines

Step 1: Collecting data

```
getwd()
```

```
## [1] "C:/Users/Madhu/Side Projects/Machine_Learning_Course"
```

```
setwd('C:/Users/Madhu/Side Projects/Machine_Learning_Course')

#Read data

letters <- read.csv("letterdata.csv")
str(letters)
```

```
## 'data.frame':    20000 obs. of  17 variables:
##  $ letter: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10
13 ...
##  $ xbox  : int  2 5 4 7 2 4 4 1 2 11 ...
##  $ ybox  : int  8 12 11 11 1 11 2 1 2 15 ...
##  $ width : int  3 3 6 6 3 5 5 3 4 13 ...
##  $ height: int  5 7 8 6 1 8 4 2 4 9 ...
##  $ onpix : int  1 2 6 3 1 3 4 1 2 7 ...
##  $ xbar  : int  8 10 10 5 8 8 8 8 10 13 ...
##  $ ybar  : int  13 5 6 9 6 8 7 2 6 2 ...
##  $ x2bar : int  0 5 2 4 6 6 6 2 2 6 ...
##  $ y2bar : int  6 4 6 6 6 9 6 2 6 2 ...
##  $ xybar : int  6 13 10 4 6 5 7 8 12 12 ...
##  $ x2ybar: int  10 3 3 4 5 6 6 2 4 1 ...
##  $ xy2bar: int  8 9 7 10 9 6 6 8 8 9 ...
##  $ xedge : int  0 2 3 6 1 0 2 1 1 8 ...
##  $ xedgey: int  8 8 7 10 7 8 8 6 6 1 ...
##  $ yedge : int  0 4 3 2 5 9 7 2 1 1 ...
##  $ yedgex: int  8 10 9 8 10 7 10 7 7 8 ...
```

Step 2: Preparing the Data

Next, we explore and prepare data in step 2.

```
# Creating training and testing data set
```

```
letters_train <- letters[1:18000, ]
letters_test <- letters[18001:20000, ]
```

Step 3, training model on data

```
# Installing  kernlab

#install.packages('kernlab')


library(kernlab)

## Warning: package 'kernlab' was built under R version 3.5.2

letter_classifier <- ksvm(letter ~ ., data = letters_train, kernel =
"vanilladot")

##  Setting default kernel parameters

letter_classifier

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 7886
##
## Objective Function Value : -15.3458 -21.3403 -25.7672 -6.8685 -8.8812 -
35.9555 -59.5883 -18.1975 -65.6075 -41.5654 -18.8559 -39.3558 -36.9961 -
60.3052 -15.1694 -42.144 -35.0941 -19.4069 -15.8234 -38.6718 -33.3013 -8.5298
-12.4387 -38.2194 -14.3682 -9.5508 -165.7154 -53.2778 -79.2163 -134.5053 -
184.4809 -58.9285 -46.3252 -81.004 -28.1341 -29.6955 -27.5983 -38.1764 -
47.2889 -137.0497 -208.1396 -239.2616 -23.8945 -10.9655 -64.228 -12.2139 -
55.7818 -10.8001 -21.2407 -11.1795 -121.5639 -33.2229 -267.3926 -81.0708 -
9.4937 -4.6577 -161.5171 -86.7114 -20.9146 -16.8272 -86.6582 -16.7205 -
30.3036 -20.0054 -26.2331 -29.9289 -56.1072 -11.6335 -5.2564 -14.8153 -4.983
-4.8171 -8.5044 -43.2267 -55.9 -214.755 -47.0748 -49.6539 -50.2278 -18.3767 -
19.1813 -97.6132 -113.6502 -42.4112 -32.5859 -127.4807 -33.7418 -30.7568 -
40.0953 -18.6792 -5.4826 -49.3916 -10.6142 -20.0286 -63.8287 -183.8297 -
57.0671 -43.3721 -35.2783 -85.4451 -145.9585 -11.8002 -6.1194 -12.5323 -
33.5245 -155.2248 -57.2602 -194.0785 -111.0155 -10.8207 -16.7926 -3.7766 -
77.3561 -7.9004 -106.5759 -52.523 -107.0402 -78.0148 -74.4773 -24.8166 -
13.2372 -7.8706 -27.2788 -13.2342 -280.2869 -32.7288 -25.9531 -149.5447 -
153.8495 -10.0146 -40.8917 -6.7333 -65.2053 -72.818 -35.1252 -246.7046 -
38.0738 -16.9126 -158.18 -184.0021 -50.8427 -28.7686 -164.5969 -97.8359 -
386.1426 -160.3188 -181.8759 -38.3648 -37.2272 -60.116 -28.2074 -53.7383 -
7.8729 -12.3159 -37.8942 -72.6434 -211.8342 -58.5023 -105.1605 -176.7259 -
685.8994 -142.8147 -159.635 -366.9437 -37.6409 -73.1357 -175.1906 -131.2833 -
```

```
41.1464 -77.8404 -57.8131 -8.6365 -251.3728 -14.0836 -36.5144 -2.2292 -6.1598
-16.8011 -26.5165 -67.19 -21.3366 -221.4815 -22.9219 -4.2616 -4.7901 -0.8263
-134.7538 -8.8843 -83.1109 -23.1019 -14.4251 -5.7337 -17.5244 -29.7925 -
23.9243 -88.9084 -28.6719 -106.0564 -16.4981 -10.6486 -7.9315 -1.5742 -
91.1706 -7.3819 -118.2628 -117.5543 -48.5606 -26.6093 -71.2968 -30.4913 -
63.5712 -279.2921 -46.3025 -50.4912 -37.9431 -21.5243 -11.6202 -134.9023 -
7.516 -5.8131 -10.1595 -13.6329 -27.0293 -25.7282 -151.8511 -39.0524 -
105.4861 -34.2434 -15.7051 -10.2304 -3.6687 -98.2094 -7.4666 -15.2668 -
75.1283 -116.5382 -16.6429 -14.9215 -55.1062 -3.0636 -8.4262 -93.6829 -
38.1162 -123.1859 -4.9078 -9.1612 -1.3077 -102.9021 -23.1138 -8.5262 -57.2623
-3.4297 -20.9579 -78.2019 -50.3741 -62.3531 -6.4908 -21.9308 -2.3736 -84.3835
-126.3997 -114.8723 -26.4109 -21.5589 -61.6405 -34.9162 -66.3243 -25.1148 -
6.7203 -4.6695 -65.3518 -39.7924 -67.3505 -36.2154 -10.9031 -62.2195 -14.9491
-24.3238 -65.0847 -4.9657 -64.2797 -278.2873 -14.6902 -13.9198 -18.2059 -
9.8972 -78.2645 -17.454 -49.5929 -55.7786 -28.7673 -15.9476 -47.531 -17.4379
-71.0516 -5.6899 -6.2519 -97.5508 -3.8196 -7.0502 -1.1238 -147.6952 -28.2018
-414.2586 -32.3275 -35.1191 -4.9605 -90.2307 -151.3409 -90.0329 -27.9491 -
42.4688 -12.5118 -26.4828 -2.0045 -62.195 -9.1662 -178.4616 -1.9406 -1.9871 -
11.3982 -0.5214 -29.6136 -35.0449 -6.7569
## Training error : 0.1335
```

The last and final step 4 of evaluating model performance

```
letter_predictions <- predict(letter_classifier, letters_test)

table(letter_predictions, letters_test$letter)

##
## letter_predictions  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R
##                  A 73  0  0  0  0  0  0  0  0  1  0  0  0  0  3  0  4  0
##                  B  0 61  0  3  2  0  1  1  0  0  1  1  0  0  0  2  0  1
##                  C  0  0 64  0  2  0  4  2  1  0  1  2  0  0  1  0  0  0
##                  D  2  1  0 67  0  0  1  3  3  2  1  2  0  3  4  2  1  2
##                  E  0  0  1  0 64  1  1  0  0  0  2  2  0  0  0  0  2  0
##                  F  0  0  0  0  0 70  1  1  4  0  0  0  0  0  0  5  1  0
##                  G  1  1  2  1  3  2 68  1  0  0  0  1  0  0  0  0  4  1
##                  H  0  0  0  1  0  1  0 46  0  2  3  1  1  1  9  0  0  5
##                  I  0  0  0  0  0  0  0  0 65  3  0  0  0  0  0  0  0  0
##                  J  0  1  0  0  0  1  0  0  3 61  0  0  0  0  1  0  0  0
##                  K  0  1  4  0  0  0  0  5  0  0 56  0  0  2  0  0  0  4
##                  L  0  0  0  0  1  0  0  1  0  0  0 63  0  0  0  0  0  0
##                  M  0  0  1  0  0  0  1  0  0  0  0  0 70  2  0  0  0  0
##                  N  0  0  0  0  0  0  0  0  0  0  0  0  0 77  0  0  0  1
##                  O  0  0  1  1  0  0  0  1  0  1  0  0  0  0 49  1  2  0
##                  P  0  0  0  0  0  3  0  0  0  0  0  0  0  0  2 69  0  0
##                  Q  0  0  0  0  0  0  3  1  0  0  0  2  0  0  2  1 52  0
##                  R  0  4  0  0  1  0  0  3  0  0  3  0  0  0  1  0  0 64
##                  S  0  1  0  0  1  1  1  0  1  1  0  0  0  0  0  0  6  0
##                  T  0  0  0  0  1  1  0  0  0  0  1  0  0  0  0  0  0  0
##                  U  0  0  2  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0
```

```
##                   V  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0
##                   W  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0
##                   X  0  1  0  0  1  0  0  1  0  0  1  4  0  0  0  0  0  1
##                   Y  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0  0
##                   Z  1  0  0  0  2  0  0  0  0  2  0  0  0  0  0  0  0  0
##
## letter_predictions  S  T  U  V  W  X  Y  Z
##                   A  0  1  2  0  1  0  0  0
##                   B  3  0  0  0  0  0  0  0
##                   C  0  0  0  0  0  0  0  0
##                   D  0  0  0  0  0  0  1  0
##                   E  6  0  0  0  0  1  0  0
##                   F  2  0  0  1  0  0  2  0
##                   G  3  2  0  0  0  0  0  0
##                   H  0  3  0  2  0  0  1  0
##                   I  2  0  0  0  0  2  1  0
##                   J  1  0  0  0  0  1  0  4
##                   K  0  1  2  0  0  4  0  0
##                   L  0  0  0  0  0  0  0  0
##                   M  0  0  1  0  6  0  0  0
##                   N  0  0  1  0  2  0  0  0
##                   O  0  0  1  0  0  0  0  0
##                   P  0  0  0  0  0  0  1  0
##                   Q  1  0  0  0  0  0  0  0
##                   R  0  1  0  1  0  0  0  0
##                   S 47  1  0  0  0  1  0  6
##                   T  1 83  1  0  0  0  2  2
##                   U  0  0 83  0  0  0  0  0
##                   V  0  0  0 64  1  0  1  0
##                   W  0  0  0  3 59  0  0  0
##                   X  0  0  0  0  0 76  1  0
##                   Y  0  1  0  0  0  1 58  0
##                   Z  5  1  0  0  0  0  0 70
```

```
agreement <- letter_predictions == letters_test$letter

table(agreement)
```

```
## agreement
## FALSE  TRUE
##   321  1679
```

Method 2: Naive Bayes Algorithm

step 1: collecting data

```
#Read data file

credit <- read.csv('credit.csv')
```

```
str(credit)
```

```
## 'data.frame':    1000 obs. of  17 variables:
##  $ checking_balance    : Factor w/ 4 levels "< 0 DM","> 200 DM",..: 1 3 4
1 1 4 4 3 4 3 ...
##  $ months_loan_duration: int  6 48 12 42 24 36 24 36 12 30 ...
##  $ credit_history      : Factor w/ 5 levels "critical","good",..: 1 2 1 2
4 2 2 2 2 1 ...
##  $ purpose             : Factor w/ 6 levels "business","car",..: 5 5 4 5 2
4 5 2 5 2 ...
##  $ amount              : int  1169 5951 2096 7882 4870 9055 2835 6948 3059
5234 ...
##  $ savings_balance     : Factor w/ 5 levels "< 100 DM","> 1000 DM",..: 5 1
1 1 1 5 4 1 2 1 ...
##  $ employment_duration : Factor w/ 5 levels "< 1 year","> 7 years",..: 2 3
4 4 3 3 2 3 4 5 ...
##  $ percent_of_income   : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ years_at_residence  : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ age                 : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ other_credit        : Factor w/ 3 levels "bank","none",..: 2 2 2 2 2 2
2 2 2 2 ...
##  $ housing             : Factor w/ 3 levels "other","own",..: 2 2 2 1 1 1
2 3 2 2 ...
##  $ existing_loans_count: int  2 1 1 1 2 1 1 1 1 2 ...
##  $ job                 : Factor w/ 4 levels "management","skilled",..: 2 2
4 2 2 4 2 1 4 1 ...
##  $ dependents          : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ phone               : Factor w/ 2 levels "no","yes": 2 1 1 1 1 2 1 2 1
1 ...
##  $ default             : Factor w/ 2 levels "no","yes": 1 2 1 1 2 1 1 1 1
2 ...
```

Step 2: Exploring the data

```
#Using sumary to check the statistics of amunt column in the dataset

summary(credit$amount)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     250    1366    2320    3271    3972   18424
```

```
str(credit$default)
```

```
##  Factor w/ 2 levels "no","yes": 1 2 1 1 2 1 1 1 1 2 ...
```

```
#Default has two levels - yes or no
#Using table to see how many were defaulted and how many were not

table(credit$default)
```

```
##
##  no yes
## 700 300
```

Steps to develop tree based classification

```
#Before creating the testing and training data, randomizing the observations

set.seed(12345)

credit_rand <- credit[order(runif(1000)),]

#checking the summary of the original data with the randomized one to notice
any substantial changes

summary(credit$amount)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     250    1366    2320    3271    3972   18424

summary(credit_rand$amount)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     250    1366    2320    3271    3972   18424
```

Splitting data to training and testing set

```
# Choosing 75% for training set and remaining 25% as the testing set

credit_train <- credit_rand[1:750,]
credit_test <- credit_rand[751:1000,]

#Looking at percentage split of the testing and training to see if
randomization went well

prop.table(table(credit_train$default))

##
##        no       yes
## 0.6826667 0.3173333

prop.table(table(credit_train$default))

##
##        no       yes
## 0.6826667 0.3173333
```

Training a model on the data

```
#install.packages('naivebayes')

library(naivebayes)
```

```
## Warning: package 'naivebayes' was built under R version 3.5.2
```

```
naive <- naive_bayes(default ~ ., data = credit_train)
```

```
naive
```

```
## ================================== Naive Bayes
==================================
## Call:
## naive_bayes.formula(formula = default ~ ., data = credit_train)
##
## A priori probabilities:
##
##        no       yes
## 0.6826667 0.3173333
##
## Tables:
##
## checking_balance         no        yes
##       < 0 DM     0.19531250 0.46638655
##       > 200 DM   0.07031250 0.05042017
##       1 - 200 DM 0.23242188 0.32352941
##       unknown    0.50195312 0.15966387
##
##
## months_loan_duration       no      yes
##                 mean 19.19531 24.34874
##                 sd   11.00031 13.21073
##
##
## credit_history         no        yes
##      critical  0.33203125 0.15966387
##      good      0.52734375 0.57142857
##      perfect   0.02539062 0.05882353
##      poor      0.08398438 0.10924370
##      very good 0.03125000 0.10084034
##
##
## purpose                          no        yes
##   business             0.08789062 0.10504202
##   car                  0.32617188 0.35294118
##   car0                 0.01171875 0.01260504
##   education            0.05468750 0.07983193
##   furniture/appliances 0.50195312 0.42016807
##   renovations          0.01757812 0.02941176
##
##
## amount       no      yes
##   mean 2987.445 3842.592
```

```
##   sd   2507.680 3420.371
##
## # ... and 11 more tables
```

step 4: Evaluating Model performance

```
naive_predict <- table(predict(naive, credit_test), credit_test$default)
naive_predict
```

```
##
##         no yes
##   no   166  29
##   yes   22  33
```

```
#calculate accuracy of the model

accuracy <- sum(diag(naive_predict))/sum(naive_predict)*100
accuracy
```

```
## [1] 79.6
```

```
 prediction <- predict(naive, credit_test)
 check <- prediction == credit_test$default
 table(check)
```

```
## check
## FALSE  TRUE
##    51   199
```

As seen, the two evaulation methods match.