ISYE6501: Introduction to Analytics Modeling: Week 1 Homework - Audit Learners

sromele

May 17, 2019

1 Question 2.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a classification model would be appropriate. List some (up to 5) predictors that you might use.

I love beer! Is not only a great drink to have while you are socializing with your friends, but it is also a great way to discover the world: would you dare to venture into the wild Baltic area with a Baltic Porter or will you discover the refreshing taste of Mexico?
A cool trick to impress your friends is to have a blind taste of beer and immediately identify the style of beer you are having. Fortunately, with a little bit of information we could correctly classify beers by looking at some attributes.
Some of the features that can be use to classify a beer are:

1. ABV, or alcohol by volume. Some styles of beer have an higher ABV than others*;
2. IBV, or international bittering units. Some styles of beer, for example Imperial IPA, have higher IBV than others**;
3. SRM, or standard reference method (i.e. a way of gauging the beer's color). Each kind of beer as a specific range of values for SRM***;
4. Original Gravity, or how much sugar is present in the wort (liquid extracted from the mashing process) before it is fermented. Each style of beer has a specific numeric range****;
5. Final Gravity, or how much sugar is left after the fermentation. As for the original gravity, each style of beer has a specific numeric range for this attribute****.

*https://www.brewersfriend.com/2017/05/07/beer-styles-abv-chart-alcohol-by-volume-ranges-2017-update/;
**https://www.brewersfriend.com/2009/01/24/beer-styles-ibu-chart-graph-bitterness-range/;
***https://www.brewersfriend.com/2017/05/07/beer-styles-srm-color-chart-2017-update/;
****https://www.brewersfriend.com/2017/05/07/beer-styles-original-gravity-and-final-gravity-chart-2017-update/;

2 Question 2.2

2.1 Question 2.2.1

Using the support vector machine function ksvm contained in the R package kernlab, find a good classifier for this data. Show the equation of your classifier, and how well it classifies the data points in the full data set. (Dont worry about test/validation data yet; well cover that topic soon.)

In order to find a good classifier, I have implemented a small for loop (5 interactions) in order to try different values of the parameter C without having to manually change every time the code (see *Listing 1* for reference.

```
# C = i * 100 gives values inside res <- 0.8639144 0.8639144 0.8623853 0.8623853 0.8639144
# C = 100^i gives values inside res <- 0.8639144 0.8623853 0.6253823 0.6636086 0.4923547
# C = 0.1^i gives values inside res <- 0.8639144 0.8639144 0.8379205 0.5474006 0.5474006
for (i in 1:5) {
  #train the model
  svm.fit <- ksvm(datamatrix[,1:10], datamatrix[,11], type = "C-svc", kernel = "vanilladot", C
      = 100^i, scaled = TRUE)

  # predict
  pred <- predict(svm.fit, data[,1:10])
  res[i] <- sum(pred == data[,11]) / nrow(data)

}
```

Listing 1: for loop for testing multiple values of C

In total I have tried 15 different models. By looking at the trained model's accuracy, I have decided to use the one with $\mathbf{C = 100}$ because the training time is smaller than most of the other models.
The model I have trained gave me an accuracy of 86.39%!

Thanks to the code provided in the homework document, I was able to find the coefficients associated with the variables. The model's equations is:

$$\begin{aligned} &-0.0010065348x_1 - 0.0011729048x_2 - 0.0016261967x_3 + 0.0030064203x_4 + 1.0049405641x_5 - \\ &0.0028259432x_6 + 0.0002600295x_7 - 0.0005349551x_8 - 0.0012283758x_9 + 0.1063633995x_10 - \\ &0.08158492 = 0 \end{aligned} \tag{1}$$

2.2 Question 2.2.2

By using the same idea as shown in *Listing 1*, I have tried several models with *kernel = polydot* and different values of the parameter C. The highest accuracy I was able to obtain is exactly equal to the one for the *vanilladot* kernel.

Moved by curiosity, I have also tried several models with the *rbfdot* kernel. The results I have obtained are much better than the other two kernels I have tried. In some instances (for example when $C = 1000$)

I was able to have an accuracy of 99.54%!
As stated in the assignment, I have trained the model on the full data set without the split in training-validation-test data. Because of this, the accuracy measurements shown are not a reliable estimate of the model's true ability of predicting new values.

2.3 Question 2.2.3

Using the k-nearest-neighbors classification function kknn contained in the R kknn package, suggest a good value of k, and show how well it classifies that data points in the full data set. Dont forget to scale the data (scale=TRUE in kknn).

As we know, in the k-nearest-neighbors algorithm it is important to correctly tune the k parameter measuring the number of neighbors to consider when classify a new point.
Because we are still not using the split into training and validation data, it is important to train our model on all point except one, otherwise the model will use the point is trying to classify as one of the neighbors.

```
# create empty matrix to store results of double loop
fit_matrix <- matrix(0, nrow = nrow(data), ncol = 20 )

# loop "i" for testing all data -1 data point each time
# loop "j" for testing different values of k each time
for (i in 1:nrow(data)){
  for (j in 1:20){
    # train the model by predicting the response variable (R1) by using all features
    knn.fit = kknn(R1~.,data[-i,],data[i,], k = j, scale = TRUE) # use scaled data
    fit_matrix[i,j] <- fitted(knn.fit)
  }
}

# transform continuous prediction in binary
fit_matrix <- ifelse(fit_matrix > 0.5,1,0)
# improve readability of columns
colnames(fit_matrix) <- c("K=1", "K=2", "K=3", "K=4", "K=5", "K=6", "K=7", "K=8", "K=9", "K=10")

# initialize variable
accuracy <- 0
# loop over my prediction in order to find the accuracy for each value of k
for (k in 1:20){
  table(data[,11], fit_matrix[,k])
  accuracy[k] <- mean(fit_matrix[,k] == data$R1)
}
```

Listing 2: for loop for testing multiple values of k

Overall, a model with $\mathbf{k = 12}$ gives us the best result. In total I have tried 20 values of k by using a for loop as shown in *Listing 2*. You can find the number of neighbors point I have used for each model and the relative accuracy in *Table 1*. As we can see, a k value of 12 or 15 give us the better results. However, we should take into account the computation time. Since this problem uses a small data set, the computation

k	Accuracy	k	Accuracy
1	81.49%	11	85.16%
2	81.49%	12	**85.32%**
3	81.49%	13	85.16%
4	81.49%	14	85.16%
5	85.16%	15	**85.32%**
6	84.55%	16	85.16%
7	84.70%	17	85.16%
8	84.86%	18	85.16%
9	84.70%	19	85.01%
10	85.01%	20	85.01%

Table 1: k values and accuracy

time of the algorithm with $k = 12$ is not really that long. For more complex data set, we should consider to use $k = 5$ since it provides an accuracy almost identical to the case $k = 12$ but with a lower computational time.

3 Question 3.1

3.1 Question 3.1.a

Using the same data set use the ksvm or kknn function to find a good classifier using cross-validation (do this for the k-nearest-neighbors model; SVM is optional)

In this problem we can try to improve the accuracy of our models by tuning two parameters: the number of *k-fold* and the number of k neighbors to take into account during classification. As the previous problems, I have used for loop to try different values of *k-fold* without manually changing the values.

```
# shuffle the data to avoid possible unwanted effects
data_shuffled <- data[sample(nrow(data)),]
# initialize variables
results <- 0

# set number of folds
k_folds <- 10
# create the different folds
folds <- cut(seq(1,nrow(data)),breaks = k_folds, labels = FALSE)
for(i in 1:k_folds){
  # find index in dataset
  index <- which(folds == i)
  train <- data[-index,]
  validation <- data[index, ]

  #train the model
  knn.fit = kknn(R1~., train, validation[,-11], k = 12, scale = TRUE) # use scaled data
  predicted <- fitted(knn.fit)
  # transform continuous prediction in binary
```

```
  predicted <- ifelse(predicted > 0.5,1,0)
  table(validation$R1, predicted)
  accuracy[i] <- mean(validation$R1 == predicted)
}
# take the average value of accuracy over all folds
avg <- sum(accuracy)/length(accuracy)
```

Listing 3: for loop for testing multiple values of *k-fold*

Table 2 shows the accuracy for some models I have trained with different parameters. As we can see, we obtain the highest accuracy when selecting **k-fold = 5** and **k = 12**. *Table 2* gives us another valuable insight: a value of *k-fold = 5* gives better performances over *k-fold = 10*. This could be due to the fact that with *k-fold = 5* we train each model on more data (20%) against the 10% when we select *k-fold = 10*.

k-fold	k	Accuracy	k-fold	k	Accuracy
5	3	84.42%	10	3	78.85%
5	5	85.42%	10	5	83.99%
5	**12**	**85.50%**	10	12	83.69%

Table 2: k values and accuracy

3.2 Question 3.1.b

Using the same data set use the ksvm or kknn function to find a good classifier splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

In this problem we are asked to split our data into different sets in order to train and test our data. We can use the code in *Listing 4* in order to do that.

```
# Assign a number, from 1 to 3, with probabilities 70\%, 15\% and 15\%
s <- sample(1:3, size = nrow(data), prob = c(0.7, 0.15, 0.15), replace = TRUE)
# take the data associated with the number one (70\%)
train <- datamatrix[s == 1,]
# take the data associated with the number 2 (15\%)
validation <- datamatrix[s == 2,]
# take the data associated with the number 3 (15\%)
test <- datamatrix[s == 3,]
```

Listing 4: Splitting our dataset

Now that we have our sets of data, we can properly train and validate our model. As for the previous problems, I have use a for loop to train multiple *support vector machines* with different kernels and different values of the C parameter.
In this particular problem, I have trained the *SVM* on the train data set and then I have tried to predict the values of my validation data set. *Table 3* shows some of the models I have trained and the accuracy of prediction based on the validation set. For almost all the cases I have tried, I was able to have a max

accuracy of 85.45% (in some instances, not shown in *Table 3*, I had a lower accuracy.

Kernel = vanilladot		Kernel = polydot	
C	**Validation accuracy**	**k**	**Validation accuracy**
100	85.45%	100	85.45%
500	85.45%	500	85.45%
1000	85.45%	1000	85.45%

Table 3: k values and accuracy

We have successfully used our training and validation data. Is now time to see how well we can do on our test data set. First off, I have re-trained the *SVM* by setting *kernel = vanilladot* and *C = 100*. Because we will use the test data, we can be sure that our model has never seen them before.
The new model has an accuracy of **88.23%!**