ISYE 6501 Homework 1

Due: May 27, 2021

1 SVM & kNN

1.1 Question 2.1

One situation in everyday life in which classification can certainly be applied, and one that has really grabbed my interest due to its volatility during the pandemic, is the stock market. The stock of individual companies can either be classified as a "buy" or a "sell" based on several (and almost too many) indicators. The reason I find the stock market so fascinating is because of the vast amount of predictors used when building or analyzing a classifier. These predictors include but are not limited to a company's: time series data (i.e. the previous day's stock price or percent change), moving average, market cap, price-earnings ratio, average volume, standard deviation, relative strength index, etc. In a classification model, each company would be a data point with these predictors and the response variable would be buy or sell. The beauty of using classifiers like SVM or logistic regression vs. kNN, is the prediction will also carry information such as how good of a buy/sell the stock is (further from the separator) and points close to the separator will indicate a riskier classification or possibly miss-classified points. What also fascinates me is even during markets of extremely volatility, such as the pandemic, classification models can perform quite well.

1.2 Question 2.2

1.2.1 Question 1.

I examined the accuracy of a SVM classifier on the credit card data with 10 predictor variables using several different values for C or λ. The accuracy depicted is using all of the data points in the credit card .txt file. It seems that the accuracy saturates starting at C = 0.01. Recall that using a lower value for C emphasizes importance on maximizing the margin compared to miss-classifying data points and a larger C value tends to want to properly classify all data points in our total error calculation (Figure 1).

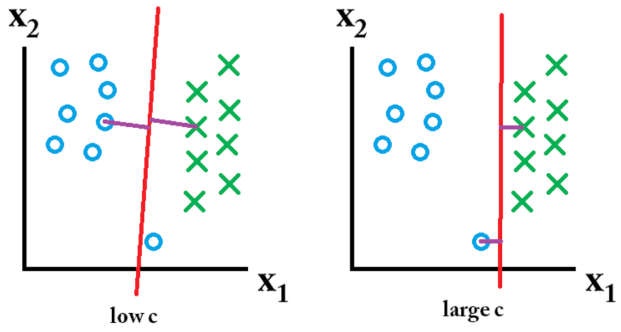

Figure 1: How tuning the C parameter trades off the between the importance of large margin vs. miss-classifying points [1].

C or λ	Model accuracy using all data points
0.0001	54.74 %
0.001	83.79 %
0.001	83.79 %
0.01	86.39 %
0.1	86.39 %
1	86.39 %
10	86.39 %
100	86.39 %
1000	86.24 %
10,000	86.24 %

Table 1: Exploring the effects of λ on SVM model accuracy

I would argue that the model accuracy saturates at a relatively low C value which could mean that the data is well separable with similar miss-classifications compared to using a separator with small margin (i.e. there isn't too much mixing of classes near the separator). I arbitrarily chose C = 1 for simplicity and found the classifier equation to be:

$$\begin{aligned} 0 = a_0 - 0.0011026642a_1 - 0.0008980539a_2 - 0.0016074557a_3 + 0.0029041700a_4 + 1.0047363456a_5 \\ -0.0029852110a_6 - 0.0002035179a_7 - 0.0005504803a_8 - 0.0012519187a_9 + 0.1064404601a_{10} \end{aligned} \quad (1)$$

Out of curiosity, I tried not scaling the data as learned in the lectures by setting scaled=FALSE, and as expected, the accuracy drops to 70.80% (see Appendix A for code).

1.2.2 Question 2.

There are other nonlinear kernels which can be used in ksvm that may improve the model's accuracy. Table 2 below shows a quick study I completed to see how different nonlinear kernels affect the classification. For consistency of comparison, C = 1 is used for all kernels. As shown, the Gaussian kernel performs the best on the data.

Kernel function	Model accuracy using all data points
Radial Basis kernel "Gaussian"	87.16 %
Polynomial kernel	86.39 %
Hyperbolic tangent kernel	72.17 %
Laplacian kernel	86.39 %
Bessel kernel	86.85 %

Table 2: Exploring the effects of nonlinear kernels on SVM model accuracy

1.2.3 Question 3.

Next, I aimed to compare the SVM results to a completely different algorithm/classification technique, kNN classification using the same credit card dataset. Figure 2 shows the results of how varying the number of neighbors being considered in classifying a new point alters the overall model accuracy when comparing all predicted data points to their true target value.

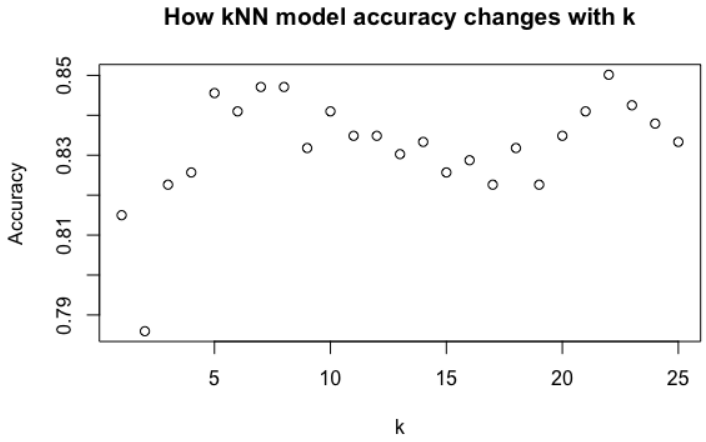

Figure 2: Varying the number of neighbors considered during classification.

The model with the highest accuracy was when k = 22 and this accuracy equated to 85.02% proving to be less accurate than the previously reported SVM model. Note that in the classification of each point for each model considered (different k), the point itself was excluded during model training so that ONLY its k neighbors were truly considered when predicting its class (see Appendix B for code). Additionally, I found [2] to be a great resource when trying to complete this task.

2 kNN with Cross-validation

2.1 Question 3.1

In the previous work, each model considered was trained using all of the data present except for the data point in which it was classifying. Now, I aim to improve the accuracy of this model analysis by appropriately splitting up the dataset into a training set, a validation set, and a test set. By training each model on a subset of the data (training set) and checking its accuracy when classifying unseen data (validation/test sets), I effectively prevent it from overfitting, being too optimistic, and therefore allow it to generalize to both real and random patterns in the data. For this problem, I use the following split (percentages of full dataset): 60% training, 20% validation, and 20% test. I chose this breakdown because the number of data points is relatively small, 654, and having too small of validation and testing sets could produce wild accuracy swings. In implementation, I followed a method explained in [3]. I further optimize each model by employing k-fold cross validation on the training set which in R yields the best choice for kernel and k (the number of evaluations during cross-validation not number of neighbors). I found these to be kernel = "inv" or inverse distance weighting and k = 16. I chose to explore k's up to 20 but no more because this would result in validation groups less than 20 data points which again can cause large accuracy swings. These optimized parameters were used in the selected model when classifying first, the validation set, and second, the test set. Figure 3 below summarizes the results of the optimized model on the validation and test sets.

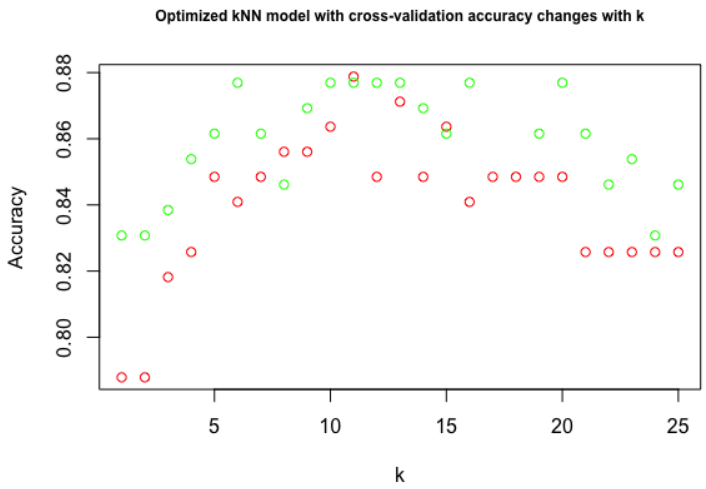

Figure 3: Varying the number of neighbors considered during classifying the validation set (GREEN) and testing set (RED).

As expected, several trends can be seen here. 1) The accuracies reported are in fact different and larger compared to Figure 2 even when using the same k values. This is because our optimized models now encompass both real and random patterns and are not overfit so they do a better job when classifying unseen data. Also, the model used the optimized parameters discovered during cross-validation. As a note, the best model accuracy in Figure 3 is 87.88%, higher than any reported in this paper. 2) The validation accuracies are almost always higher for a given k. This precisely why validation is completed because there is selection bias; some random patterns present could match what the model was trained on and the accuracies are therefore overly optimistic. The unbiasing occurs when we then utilize the test set, additional unseen data that is different from the validation set. 3) Trends in the validation and testing accuracies roughly align. Recall, another reason validation is done is to hone in on a specific model choice and then to extensively analyze it further. So one would hope that high validation accuracies actually indicate that the testing accuracy will be high too and vice versa. As additional discussion, it is worth mentioning that the models here could be retrained using the training and testing set but then there is a trade off between model quality and unbiasing.

References

[1] https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

[2] https://rstudio-pubs-static.s3.amazonaws.com/349520_6c62f724297f4084abb48493c6f703a5.html

[3] https://stackoverflow.com/questions/36068963/r-how-to-split-a-data-frame-into-training-validation-and-test-sets

Appendix A SVM code

```
#install.packages("kernlab")
library(kernlab)
data <- read.table("credit_card_data.txt")
data <- as.matrix(data)

# calling svm, using vanilladot as simple linear kernel
model <- ksvm(data[,1:10], data[,11], type="C-svc", kernel="vanilladot", C=1, scaled=TRUE)

# calculate a_1...a_m
a_vector <- colSums(model@xmatrix[[1]] * model@coef[[1]])

# calculate a_0
a0 <- model@b
pred <- predict(model, data[,1:10])
#print(typeof(pred))

# frac of model predictions match actual classification
acc <- sum(pred == data[,11]) / nrow(data)
#print(a_vector)
```

Appendix B kNN code

```
#install.packages("kknn")
library(kknn)
data <- read.table("credit_card_data.txt")

acc_array <- vector()
for (k in 1:25) {
  pred_knn <- vector()
  #print(pred_knn)
  #print(dim(data)[1])
  for (i in 1:nrow(data)) {
    model = kknn((data[-i,11])~., train=data[-i,1:10], test=data[i,], k = k, distance = 2,
                 kernel = "rectangular", scale=TRUE)
    pred_knn[i] <- predict(model)

    pred_rounded <- as.integer(round(pred_knn))
  }
  #print(k)
  acc_array[k] <- sum(pred_rounded == data[,11]) / nrow(data)
}

plot(acc_array, main='How kNN model accuracy changes with k', xlab='k', ylab='Accuracy')
```

Appendix C kNN cross-validation code

```
#install.packages("kknn")
library(kknn)
data <- read.table("credit_card_data.txt")

# splitting data
train_size <- floor(0.6 * nrow(data))
val_size <- floor(0.2 * nrow(data))
test_size <- floor(0.2 * nrow(data))

# random sampling of data w/o replacement
train_idx <- sort(sample(seq_len(nrow(data)), size=train_size))
not_train_idk <- setdiff(seq_len(nrow(data)), train_idx)
val_idx <- sort(sample(not_train_idk, size=val_size))
test_idx <- setdiff(not_train_idk, val_idx)
#test_idx <- sort(sample(not_train_idk, size=test_size))

# different sets
train <- data[train_idx,1:10]
val <- data[val_idx,1:10]
test <- data[test_idx,1:10]

model = train.kknn((data[train_idx,11])~., data=data[train_idx,1:10], kmax= 20,
                   distance = 2, kernel = c("optimal","rectangular", "inv", "gaussian",
                    "triangular", "cos", "biweight", "triweight"), scale=TRUE)

#print(sum(as.integer(round(predict(model, test))) == data[test_idx,11]) / nrow(test))
print(model)
acc_array <- vector()
acc_array_test <- vector()
# this is run for val and test
for (k in 1:25) {
  model = kknn((data[train_idx,11])~., train=data[train_idx,1:10],
               test=data[test_idx,1:10], k = k, distance = 2, kernel = "inv", scale=TRUE)

  pred_rounded <- as.integer(round(predict(model)))
  acc_array_test[k] <- sum(pred_rounded == data[test_idx,11]) / nrow(test)
}

plot(acc_array_test, main='Optimized kNN model with cross-validation accuracy
     changes with k', xlab='k', ylab='Accuracy', col="red", cex.main=0.75)
points(acc_array, col="green")
```