Georgia Tech

Homework 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.

A situation from my job that a classification model would be appropriate would be an email spam detection. Since I get a lot of emails in my day-to-day work life, sometimes I cannot tell if they are real or fake, and the best situation I can do is just to ignore them. Sure, there is a button to mark as spam, but that alerts everyone who gets an email, which is a safe option. Some possible predictors I may use would be: frequency of certain keywords, sender reputation score, number of links in the email, if there are any attachments, and the way the email is formatted.

Question 2.2

The files `credit_card_data.txt` (without headers) and `credit_card_data-headers.txt` (with headers) contain a dataset with 654 data points, 6 continuous and 4 binary predictor variables. It has anonymized credit card applications with a binary response variable (last column) indicating if the application was positive or negative. The dataset is the "Credit Approval Data Set" from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Credit+Approval) without the categorical variables and without data points that have missing values.

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. (Don't worry about test/validation data yet; we'll cover that topic soon.)

   Notes on `ksvm`

   - You can use `scaled=TRUE` to get `ksvm` to scale the data as part of calculating a classifier.
   - The term λ we used in the SVM lesson to trade off the two components of correctness and margin is called C in `ksvm`. One of the challenges of this homework is to find a value of C that works well; for many values of C, almost all predictions will be "yes" or almost all predictions will be "no".
   - `ksvm` does not directly return the coefficients a_0 and $a_1 \ldots a_m$. Instead, you need to do the last step of the calculation yourself. Here's an example of the steps to take (assuming your data is stored in a matrix called data):[1]

```
# call ksvm.  Vanilladot is a simple linear kernel.
model <- ksvm(data[,1:10],data[,11],type="C-
svc",kernel="vanilladot",C=100,scaled=TRUE)
# calculate a1…am
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
```

[1] I know I said I wouldn't give you exact R code to copy, because I want you to learn for yourself. In general, that's definitely true – but in this case, because it's your first R assignment and because the ksvm function leaves you in the middle of a mathematical calculation that we haven't gotten into in this course, I'm giving you the code.

```
a
# calculate a0
a0 <- -model@b
a0
# see what the model predicts
pred <- predict(model,data[,1:10])
pred
# see what fraction of the model's predictions match the
actual classification
sum(pred == data[,11]) / nrow(data)
```

Hint: You might want to view the predictions your model makes; if C is too large or too small, they'll almost all be the same (all zero or all one) and the predictive value of the model will be poor. Even finding the right order of magnitude for C might take a little trial-and-error.

Note: If you get the error "`Error in vanilladot(length = 4, lambda = 0.5) : unused arguments (length = 4, lambda = 0.5)`", it means you need to convert `data` into matrix format:

```
model <-
ksvm(as.matrix(data[,1:10]),as.factor(data[,11]),type="C-
svc",kernel="vanilladot",C=100,scaled=TRUE)
```

I. Methodology

1. First, I will load and prepare the data to make sure there are indeed 654 observations, 10 predictors, 6 of them being continuous and 4 of them being binary, as shown in the code file. Next, I will split the dataset into X and Y, X being the predictor variable of the first 10 columns, and Y being the last column as the response variable. I convert X into a matrix because KSVM may require matrix input, and I convert Y into a factor because this is a classification problem (binary classes 0/1). The next step would be to train a linear SVM classifier using KSVM. Importing the kernlab function KSVM with:
   - type = "C-svc" (classification SVM)
   - kernel = "vanilladot"
   - scaled = TRUE (so the predictors are automatically scaled)
   - C = 10 (the cost parameter that controls the tradeoff between margin size and classification errors)

II. Results

Since KSVM does not output the equation $a_0 + a_1x_1 + \cdots + a_{10}x_{10}$, I computed it manually by calculating the coefficient vector $a_1 \ldots a_{10}$ using support vectors and their coefficients. This was the formula I used, a=∑(support vectors * model coefficients). Computing the intercept a_0 = -b and the classifier output came out to
f(x)= 0.08157559 - 0.0009033671A1 - 0.0007891036A2 - 0.0016972133A3 + 0.0026113629A8 + 1.0050221405A9 - 0.0028363016A10 - 0.0001569285A11 - 0.0003925964A12 - 0.0012784443A14 + 0.1064387167A15

My decision rule was if f(x) was > to 0, then it would be positive as a predict 1, and vice versa for predict 2. My final step was using predict(model,X) to generate predictions for all 654 and to compute accuracy. Having an 86.4% accuracy, correctly classifying 565 out of 654 data points.

III. Discussion of results

The linear SVM achieved 86.4% accuracy, getting 565 out of 654 applications right. That's above average for just a linear decision boundary. The model found some useful patterns in the data for separating approved from rejected applications. What's interesting is how unevenly the features contribute. Looking at the coefficients, A9 is the highest, having a way more influence on the classification than most other variables. This suggests A9 could mean something, at least according to this dataset. The fact that a simple linear model works this well tells us the predictor variables have real signals. There's meaningful structure here that even a straightforward approach can pick up on.

2. You are welcome, but not required, to try other (nonlinear) kernels as well; we're not covering them in this course, but they can sometimes be useful and might provide better predictions than `vanilladot`.

In addition to the linear kernel, I also explored a nonlinear support vector machine using the Radial Basis Function (RBF) kernel. The RBF kernel allows the classifier to construct a nonlinear decision boundary, which can be useful when the relationship between predictors and the response variable is more complex and not well separated by a linear boundary.

IV. Discussion of RBF Kernel Results

Using a nonlinear Radial Basis Function (RBF) kernel, the support vector machine achieved a training accuracy of 91.4%, correctly classifying approximately 598 out of 654 credit card applications. This has a higher improvement over linear SVM, which achieved an accuracy of 86.4% on the same dataset. The improvement in accuracy is expected, as the RBF kernel allows for a nonlinear decision boundary, enabling the model to capture more complex relationships among the predictor variables. By implicitly mapping the data into a higher-dimensional feature space, the RBF kernel can separate observations that are not linearly separable in the original input space. As with the linear model, the predictors were automatically scaled to ensure that all variables contributed appropriately to the distance-based calculations used by the RBF kernel.

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. Don't forget to scale the data (`scale=TRUE` in `kknn`).

   Notes on `kknn`

   - You need to be a little careful. If you give it the whole data set to find the closest points to i, it'll use i itself (which is in the data set) as one of the nearest neighbors. A helpful feature of R is the index `-i`, which means "all indices except i". For example, `data[-i,]` is all the data except for the ith data point. For our data file where the first 10 columns are predictors and the 11th column is the response, `data[-i,11]` is the response for all but the ith data point, and `data[-i,1:10]` are the predictors for all but the ith data point.

(There are other, easier ways to get around this problem, but I want you to get practice doing some basic data manipulation and extraction, and maybe some looping too.)

- **Note** that `kknn` will read the responses as continuous, and return the fraction of the k closest responses that are 1 (rather than the most common response, 1 or 0).

V. Methodology

1. To avoid a data point being classified using itself as one of its nearest neighbors, I removed the i-th observation from the training data when predicting the i-th response. This was done using the index -i, so that each prediction was made using all observations except the one being classified.
2. The KKNN function returns the fraction of the k nearest neighbors whose response is 1, rather than a direct class label. I converted this value into a classification rule by predicting 1 if the fraction was at least 0.5, and otherwise 0.
3. I tested several values of k and found that k = 7 has a good balance between sensitivity to noise and over smoothing. Using this value of k, I computed predictions for all 654 data points and calculated the overall classification accuracy.

VI. Results

A. Using k = 7, the KNN classifier achieved an accuracy of approximately 84.7% correctly classifying 554 out of 654 observations. This performance is comparable to the linear SVM classifier but lower than the nonlinear SVM with an RBF kernel.

VII. Discussion of Results

The KNN classifier performs reasonably well on this dataset, demonstrating that local similarity among observations contains useful information for predicting credit approval outcomes. However, its accuracy is lower than both SVM models considered, suggesting that KNN is less effective at capturing the global structure of the data. One limitation of KNN is its sensitivity and the choice of k. Smaller values of k can lead to high variance, while larger values may oversmooth the decision boundary and reduce accuracy. Additionally, KNN does not produce an explicit classifier equation, making it less interpretable than the linear SVM. Compared to the nonlinear SVM with an RBF kernel, KNN performs at a less accurate value, indicating that a flexible decision boundary is better suited for this dataset. As with the previous models, the accuracy reported here reflects training performance, and further evaluation using validation or test data would be necessary to assess generalization.