Name: Zack Burch

Course: CSC 570AL

Assignment: Final Project

Date: 12/2/2016

Final Project Report

Describing What I Have Done

For this project, I have created two models – a decision tree and a support vector machine – that can be used to predict the party of a congressman based on voting record.

In preparing the data, I added column names representing the vote that took place, to aid readability. From there, I imputed the missing values. This involved creating a cross table between the votes themselves and the party the congressman belonged to. If democrats largely voted yes, I checked to see if the unknown vote was given by a democrat, if it was, then I imputed a "yes" vote, otherwise, I imputed "no". The same logic also captured republican votes, so that if republicans largely voted yes, I imputed a "yes" vote if the congressman was a republican. There was one case where both democrats and republicans largely voted yes, and "yes" was imputed for every unknown vote. By the end of this exercise, all votes for all columns were filled with either "yes" or "no". There were no unknown votes remaining.

For both models, I followed a four step pattern:

1) Building the model manually (not using the caret package) using the packages outlined in the book and lecture. They produced fairly barebones models that were not very robust – single parameter set and no cross-validation or ensemble learning. In this step, during the decision tree model build, I defined the test and training datasets for both models. I chose them at random via the "sample" method, and compared the proportions of republicans and democrats for the train and test data against the proportions in the entire dataset to ensure that train and test were accurate reflections of the full dataset. For the decision tree model I used the C5.0 algorithm from the C50 package. For the SVM model I used the ksvm algorithm from the kernlab package.
2) Running 10-fold cross-validation on these same models to assess how well the barebones models will generalize. I used the caret package to create the folds, but used the same algorithms from the first step to create the models. I then took the mean kappa statistic across all folds to determine how well the model will generalize.
3) Perform parameter tuning to provide a broader range of potential models that would identify the most accurate model from several possibilities. For the decision tree model, I chose to tune the "trials" and "winnowing" parameters. The model remained "tree" for all iterations. While the lecture and the book kept "winnowing" set to FALSE, I decided to see whether allowing the algorithm to remove attributes it deemed unimportant from this particular dataset would have any positive impact on the end model.
4) Finally, I tried to improve upon that model by using ensemble learning. I chose Bagging for the ensemble learning technique for both models, in order to compare the effects of bagging in two

different types of models. I used the “bagging” method from the ipred library to perform bagging on the decision tree, and I used the bagControl() method from the caret library to perform bagging for the SVM.

I tried to keep the process for both models as similar to one another as possible, to provide the most accurate comparison between the two final models.

The largest difficulty I faced in this project was determining the best method for imputing the values into the “?” field. Each method I considered prior to the steps I chose, such as putting the value that represented the majority of votes for a single attribute, seemed as though it would result in nearly as many errors as it did true cases. In that scenario, imputing values would not result in any more accuracy than leaving the “?”. Once I decided on the method of tying the value for the imputation to the direction the party they are classified as largely voted, I became confident in my choice and was able to move forward. From that point, the most difficult part was translating what I wanted to do into R code. I’m certain I could do what I did in a more elegant and efficient way, but to ensure I had enough time to finish the rest of the project, I just wrote the code for each attribute on its own.

The algorithms were fairly straightforward to write, and it was an excellent exercise going back to decision trees and SVMs, after reading the rest of the book and gaining new perspectives. I have a much better understanding of both after completing this project.

Algorithm Comparisons

The Decision Tree algorithm produces a model that is fairly easy to follow, especially compared to a Support Vector Machine. It creates many splits on the data that provide a directional map based on values of certain attributes. The order in which the splits occur in the tree depends largely on which splits will result in the lowest entropy – which is a measure of disorder. The lower the entropy, the lower the disorder. By processing splits in this way, we typically get a much shorter tree than if we tried to create the tree using an arbitrary split-order. At the end of the model creation, we can follow the exact decision points so that if someone asked us, “why was this person classified as a Republican?”, we can work through the exact logic to where they were determined to be republican.

While the decision tree produces a very transparent model, a support vector machine (SVM) is a black box method – meaning it is very difficult to determine, much less explain, how a particular classification was determined. An SVM attempts to divide the data linearly into dimensions via hyperplanes. In cases where the data cannot be separated linearly, there are a couple “tricks” the algorithm can use. One is the use of slack variables, which are used to penalize variables that are aligned in an incorrect dimension. This penalization will allow the algorithm to optimize to find the lowest feasible penalization score across all values. Another way to account for non-linearly separable data is with what is called “kernel tricks”. Kernel tricks transform the data in a way that makes differences in the data more clearly defined. An SVM can model some very complex data, and if the classification reasoning doesn’t have to be explicitly defined, an SVM can be a great choice.

To summarize, decision trees are great and versatile learners, especially when accountability is necessary. SVMs are excellent learners that can find very interesting and mathematical relations in the data, but will not allow it’s reasoning to be easily seen or communicated.

Summarizing the Results

To summarize the results, I will first compare each model to the other in regards to the steps outlined in the first section. Then, I will compare the best performing model of the decision tree to the best performing model of the SVM.

1) Manual build of the model.
    a. The decision tree produced a model with an accuracy of 94.3% (82 of 87). There were 4 democrats classified as republicans, and 1 republican classified as a democrat. Given that there are many more democrats in our dataset than there are republicans, this result makes sense.
    b. The support vector machine fared slightly better with an accuracy of 95.4% (83 of 87). There were 3 democrats classified as republicans, and 1 republican classified as democrat.
2) Evaluating future performance
    a. Upon running 10-fold Cross-Validation on the decision tree model created in the first section I found the mean kappa across all 10-folds was .9377 – suggesting excellent model performance against future data.
    b. Running 10-fold Cross-Validation on the SVM model in the first section resulted in a kappa value of .9374. This is nearly as good as the decision tree model.
3) Perform automated parameter tuning
    a. For the decision tree model, assigning various values to trials and allowing both true and false for winnowing, the best model resulted from 15 trials, and Winnowing set to FALSE. This gave an accuracy of 97.71% - definitely better than the 94.3% accuracy found in the first step. The kappa value also jumped – it came in at .9522 – well above the .9377 found using manual 10-fold CV.
    b. The tuning of the cost variable resulted in a model that benefited most from a cost of 1 – leaving an accuracy of 97.03%, and a kappa value of .9374. This kappa value remained unchanged from when I ran the manual 10-fold CV.
4) Try to improve with ensemble learning
    a. The decision tree model that was created using bagging produced a model that tested exceptionally well against the existing dataset – it only misclassified 1 case. This is to be expected, however, since bagging uses a very large portion of the total dataset to train the algorithm. When trying to see how well tree bagging will generalize, it appears it doesn't do so well. Using the "treebag" method in the train function of the caret library, we see that kappa results in only .9178. While any kappa above .9 is excellent, the learner we built in the previous section resulted in a model that will generalize much better.
    b. Bagging with the SVM resulted in a similar decline of kappa, but not to the extent that the decision tree declined. In the SVM – we generated a kappa of .9376.

Overall, considering everything, I would choose the parameter tuned decision tree model with 15 trials, and no winnowing. This model produced the highest accuracy and kappa. This alone may not be enough to go by, but given that the decision tree learner produces a transparent model that enables us to see why a classification occurred, it seems this would be the most useful model going forward.