|  |
| --- |
| CSC 570 AL – Machine Learning |
| Machine Learning in Politics |
| Spring 2017 – Final Project |
|  |
| **Justin Orr** |
| **5/3/2017** |

Introduction

Machine learning provides sophisticated functionality to enable people to understand data and help predict outcomes of different scenarios based on previous knowledge. It is a powerful tool that provides insight where previously undiscovered. Using a dataset which includes 16 key votes for 435 U.S. House of Representatives Congressmen of the 98th Congress, 2nd session, 1984, two machine learning algorithms (K Nearest Neighbors (k-NN) and C5.0) can be taught to determine which representatives are democrats and which are republicans given only the associated voting data. Then using different methods to tune the algorithms it can be determined exactly what values provide the best results.

Building the Dataset

The first and most crucial steps with any machine learning algorithm are the processes of importing the data and preparing it to be analyzed. The house voting records come stored as a table in a text file called “house-votes.txt”. Each row of the table contains 17 categorical variables. The first of which is the party name, denoted by either “democrat” or “republican”, and is followed by 16 recorded votes made by each individual with values of either yes (“y”), no (“n”), or abstained from voting (“?”). Given that ahead of time, it is known how the dataset is formatted; this is a good time to define a vector which contains labels for each of these pieces of information. Going forward, each variable can be referenced independently with knowledge of what data lies within. To do so, a vector is created to hold these label names:

**votercols <- c( "Party", "HI", "WPCS", "AOTBR", "PFF", "ESA", "RGIS", "ASTB", "ATNC", "MxM", "Imm", "SCC", "ES", "SRTS", "Crm", "DFE", "EAASA" )**

Now that these labels have been prepared, the text file can be read into a data frame:

**voters = read.table("house-votes.txt", header = F, sep=",", stringsAsFactors=T, col.names=votercols)**

In this method, parameters include the name of the text file, whether or not header data is present, how the data is delimited, whether or not each value should be treated as a categorical factor, and the predefined column names.

The voting data is housed within a data frame, but there is a problem; all of the data is categorical. To run the voting information within the selected machine learning algorithms, it needs to be converted into numeric values. A secondary issue is revealed, in which all of the votes contain “missing” data denoted by the “?” representing abstained votes. In order to use the data, the values must go through the imputation process. This involves filling in the missing data with a “guess” value or dummy coding the missing values. For example, the first column’s voting information (handicapped-infants; denoted as “HI”) looks like:

**$ HI : Factor w/ 3 levels "?","n","y": 2 2 1 2 3 2 2 2 2 3 ...**

To dummy code this, it can be converted into two separate numeric variables: HI.YES and HI.NO. HI.YES has a value of 1 for every cell which contains the value “y” and HI.NO has a value of 1 for every cell which contains the value “n”:

**HouseVotes$HI.YES <- ifelse(HouseVotes$HI == "y", 1, 0)**

**HouseVotes$HI.NO <- ifelse(HouseVotes$HI == "n", 1, 0)**

With these values set as so, any abstained votes will now become 0 for both variables. This step is repeated for all 16 vote columns and are added to the same “voters” data frame the data was initially read into. Once complete, a new data frame can be created by selecting the first column (party affiliation) and the 32 newly created numeric columns:

**HouseVotesData <- HouseVotes[c(1,18:49)]**

Running the k-NN and C5.0 Algorithms

With the data fully prepared, both machine learning algorithms can process the information. The first algorithm is k-Nearest Neighbors, for which a training and test data set must be created from the “HouseVotesData” data frame. This is accomplished by utilizing all of the columns except the first from the data set. The training set should be significantly larger to provide more data points to learn from, in turn causing the model to be more accurate. In this example, the test set contains 100 voting records, and the training set is the remaining 335. Training and test label vectors will be created within the same data using only the first column.

**HouseVotesTrainKNN <- HouseVotesData[1:335, -1]**

**HouseVotesTestKNN <- HouseVotesData[336:435, -1]**

**HouseVotesTrainLabels <- HouseVotesData[1:335, 1]**

**HouseVotesTestLabels <- HouseVotesData[336:435, 1]**

Now the k-NN algorithm can be used to predict the values of the 100 test records. A ‘K’ value of 10 is used as it is half of the square root of the total number of voting records:

**KNNPredict <- knn(train=HouseVotesTrainKNN, test=HouseVotesTestKNN, cl=HouseVotesTrainLabels, k=10)**

Now that the k-NN algorithm has predicted 100 party affiliations, the C5.0 algorithm can be used similarly so that the results of both predictions can be compared. The C5.0 algorithm requires only a test data set. As performed previously, the last 100 vote records are stored in separate vectors. The C5.0 algorithm uses the full data set to train a model, then uses this model against a test data set, which predicts the party affiliation:

**HouseVotesTestC50 <- HouseVotesData[336:435, -1]**

**HouseVotesTestLabels <- HouseVotesData[336:435, 1]**

**C50Model <- C5.0(HouseVotesData[-1], HouseVotesData$Party, trials = 10)**

**C50Predict <- predict(C50Model, HouseVotesTestC50)**

Now that both algorithms have made predictions about the test dataset, the results of both can be compared using confusion matrices. An accuracy value of 97% was established for the C5.0 algorithm, and an accuracy value of 89% was found for the k-NN algorithm. These accuracy values are both high, but require further comparison using the Kappa Statistic. The Kappa Statistic adjusts accuracy by accounting for the possibility of a correct prediction simply by choosing the most frequent class. The Kappa value for the C5.0 algorithm is 93%, which is considered ‘Very Good’, while the k-NN algorithm has a Kappa value of 77.6%, which is only considered ‘Good’. Lastly are the Sensitivity and Specificity values; the ability to classify True positives and True Negatives correctly, respectfully. For C5.0, the Sensitivity is 100% and the Specificity is 95%; both extremely good but somewhat to be expected, considering the Accuracy value. For k-NN, the Sensitivity was 92% and the Specificity is 86%.

Estimating Performance with 10-Fold Cross Validation

Running each algorithm against a single test case is standard for initially designing and testing your data set and parameters, but doesn’t necessarily provide an indication of how well it will perform on other similar datasets. Testing methodology known as k-fold cross validation exists for this purpose. K-Fold CV, or in this scenario 10-Fold, randomly divides the data set into ‘k’ separate partitions called ‘folds’ and uses each ‘fold’ as a single test case against the other 9 folds. Using this method, a single data record is never used more than once, making each test case unique. In order to use 10-Fold CV for either algorithm, the first step is to split the data set into ‘k’ folds:

**folds <- createFolds(HouseVotesData$Party, k=10)**

This splits the entire house voting record dataset into 10 evenly filled datasets. Now, a function needs to be created which can take in each of the test ‘folds’ and run the C5.0 algorithm for each one:

**C50CVResults <- lapply(folds, function(x) {**

**C50Train <- HouseVotesData[-x,]**

**C50Test <- HouseVotesData[x,]**

**C50Model <- C5.0(Party ~ ., data = C50Train)**

**C50Predict <- predict(C50Model, C50Test)**

**C50Actual <- C50Test$Party**

**kappa <- kappa2(data.frame(C50Actual, C50Predict))$value**

**return(kappa)**

**})**

The results vector now contains the resulting Kappa statistic for each test fold. Averaging these Kappa statistics results in a final value of .91. According to the accepted range of Kappa statistics, a value of .91 is in ‘Very Good Agreement’ between the model’s predictions and the true values of our Party column. Repeating the same steps for the k-NN algorithm provides a result of .85 for the averaging Kappa statistic.

Automated Parameter Tuning

Though a significant number of tests have been run against the data set, there are a few variables left to the process of guess-and-check. Automated parameter tuning is a process which conducts a search through many possible parameter values in an attempt to find the ones that best fit. Setting up automated parameter tuning to be used with the C5.0 algorithm is simple. Data required includes the dataset to be trained from, the label vector containing the true values, and the name of the method to be tuned:

**C50ParameterModel <- train(Party ~ ., data=HouseVotesData, method ="C5.0")**

This builds a model with a summary containing a list of all of the evaluated candidate models and a suggested choice that provides the highest accuracy. Using this model against the full house votes dataset should produce the most accurate results from the C5.0 algorithm:

**C50ParameterPredict <- predict(C50ParameterModel, HouseVotesData)**

**table(C50ParameterPredict, HouseVotesData$Party)**

Using the parameter tuned model against the dataset produces a prediction with an accuracy rate of 98.9%.

Using the same steps for the k-NN algorithm returns a model with suggested values of ‘k’, which is the only parameter in the k-NN algorithm which is ‘tune-able’. Using the suggested model created from the train method produces a prediction which provides an accuracy rate of 94.3%.

Improving with Ensemble Learning

To this point, much of the algorithm testing and tuning has been completed independently. It is possible to combine all of these tuning processes into one strong learning team, called an ensemble learner. The first task to complete when preparing to create an ensemble learner is to decide which to use. For the scenario, a Random Forest approach will be taken because it combines some of the best features of other ensemble learners. The only down side is the ease of interpreting the resulting data.

Preparing to use the Random Forest approach begins by creating a set of control options in which the algorithm determines which type of tests to run and how many times:

**ctrl <- trainControl(method="repeatedcv", number=10, repeats=10)**

A repeated cross validation method will be used with a ‘fold’ value of 10; which will be repeated 10 times to ensure correct averages.

Next, a tuning grid must be created in order to tell the ensemble learner what values to use for which parameters. Since the only parameter in the k-NN algorithm that is ‘tune-able’ is the ‘k’ parameter, the grid must be told which values of ‘k’ to try:

**KNNGrid <- expand.grid(.k=c(2,3,4,5,6,7,8))**

Finally, combining the control and grid options with the Random Forest training method results in the final model:

**KNNRFModel <- train(Party ~ ., data = HouseVotesData, method="knn", metric = "Kappa", trControl=ctrl, tuneGrid=KNNGrid)**

From this model’s summary, it is shown that the most accurate value of k is a value of 3. It provides both the highest accuracy value and the highest Kappa statistic. Following all the same steps for the C5.0 algorithm results in learning that running 20 trials with the model parameter set to ‘tree’ and the winnow parameter set to ‘FALSE’ results in the highest accuracy and Kappa statistic.

Conclusion

Designing, preparing, and tuning machine learning algorithms can be a time consuming and complicated endeavor, but the amount of power and knowledge that can be gained from a finely setup algorithm is immeasurable. Throughout this project, two machine learning algorithms were used against the house vote dataset to demonstrate how the preparation process takes place. In conclusion, all of the appropriate parameters and values were discovered for both algorithms. It was determined that the C5.0 algorithm performs exceptionally better that the k-NN algorithm on this particular dataset, as evidenced by both higher accuracy values and the confirmed Kappa Statistics at each stage of testing.