Austin Prince

Data Mining

10/26/17

In this project I am using Data Mining techniques to analyze the relevant genes in Alzheimer’s (AD) patients. I have two datasets for this project one containing patients with AD and control dataset of patients who do not have AD. These datasets contain patients and a record of the gene expression levels of 8000 genes in these patients. In the AD patient data set there are 176 patients and in the control dataset there are 188 patients, both of these datasets are matched up in that they contain the gene expression levels for the same 8000 genes in these patients

**Preprocessing**

In order to preprocess my data I translated the *.gex* files which were given to us and put them into an Excel spreadsheet where I could save them as *.csv* files which are much easier to work with in Python, the language I will be using for this assignment. In addition to this I selected only the first 1800 genes from the dataset to analyze in order to reduce the computational time that would be needed to construct the random forest and SVM models.

**Imputation**

The first thing I had to do to this dataset was to account for missing values in the data. In both datasets there were a significant amount of missing values. These showed up as *NA* in the *csv* files. Before I did any analysis with this dataset I had to fill in these values. I decided to use a Gaussian distribution to fill in the values of my missing data. I describe the process in pseudocode below.

translate .csv file into a python DataFrame where it can be easily manipulated

transpose the DataFrame matrix such that the genes (features) are now the columns and the patients (instances) are the rows, this will make RandomForest construction easier in the future

for each column (columns represent features/genes) calculate the mean and var of gene expression for all values that exist (not NaN)

for each value in column

if value is NaN

generate a random number within previously calculated Gaussian distribution and assign it to this point

Using this method of imputing missing data based on a normal distribution we are assuming that our gene expression in patients follows a normal distribution and then randomly generating a data point within that distribution. If there were more instance data, in the form of more patients then the values of the imputed values of the data would be closer to an actual value. Another way to do this would be to use a kNN algorithm in this case but I figured that the ease of computing using a Gaussian distribution outweighed the potentially more accurate kNN estimation. In addition to this I also supposed that given the amount of data points would be enough such that a misestimating of the value of one data point using random numbers in a Gaussian distribution would be insignificant in the long run.

**Random Forest Feature Selection**

In order to select the genes that are most significant in the development of AD patients I constructed a random forest consisting of 500 trees for each set of frequency and instance percentages specified. I used Python’s scikit learn package to construct the trees and I will briefly describe the pseudocode for constructing a random forest below.

function RandomForest

for i in range num\_trees

A = bootstrap sample from training set

Ta = treeLearn(A)

add Ta to set of trees T

return set of trees T

treeLearn(D)

at each node

cycle through all features in D and all possible splits of features in D

split on best feature (highest information gain)

continue until all data is uniformly classified or pruning criteria is met

return tree

I used the RandomForestClassifier that is available in scikit learn in Python for this code. This model takes in several parameters that specify how the RandomForest classifer should be constructed.

n\_estimators – takes in the number of trees that should be constructed in the RandomForest

max\_features – the percent of features from the data set that should be used to construct the RandomForest

criterion – this is the criteria that the nodes in the individual decision trees should split on, for example in the C4.5 algorithm decision trees split on the node that has the highest gain ratio (GR) and for ID3 it is the ndoe that has the highest information gain, I used the criterion entropy

Because a random forest introduces randomness into the construction of decision trees it is a good tool to overcome common problems such as overfitting and bias towards certain features over others.

1. The decision tree selects attributes based on their information gain, or how important they are in discriminating between the two target classes. The Random Forest constructs an ensemble of these decision trees by randomly selecting certain features. If a gene (feature) is split on more often this means that it is more descriptive in discriminating between the two classes (AD patients and non-AD patients). Because the genes that are selected to construct the RF are chosen randomly from the whole set this means that they all have the same probability of appearing in the RF. Because of this we can say that the features that the decision trees in the Random Forest split on more often are more important in determining whether a patient has AD or not. Using the above conclusion, we can say that if a gene appears more frequently (higher frequency) then this gene will be a more important feature (provides more information on which class this instance will be). One aspect of the RF that I did not take into account was the fact that the first feature that is split on is usually the most descriptive. The root node of a tree in a decision tree is the most descriptive overall in discriminating between the two classes, more descriptive that the splits further down in the tree. I neglected to take this into account, although given that I used a RF with 500 trees I assumed that the randomness of this process would allow the genes that are the most important to simply occur more often and this would provide the necessary information. For example the most important gene would always be selected as the root node for trees in the RF and for my testing in which I built a RF containing 500 trees over the course of 18 trials the number of occurrences of these genes in the RF would signify their importance. The process of extracting which features were used in the random forest to split on is described in the following pseudocode.

function getFeatures()

for trees in RandomForest

for each split in tree

add to list featureSplit

calculate frequency of features in RF based on how many occurences of feature in the list featureSplit

For this Random Forest test I constructed 18 Random Forest models using a different amount of features and instances from the test data for each tree. In the initial processing of the data I selected only the first 1800 genes from the model (out of 8000 total genes). For the individual RF models I fixed the features (genes) for the RF at 70% and then constructed trees for 50 to 90% of instances with a step size of 5%. I did the same thing with instances and features, fixed instances (patients) at 70% and varied features (genes) by 5% from 50% to 90%. In my preprocessing I set up my Data Frame so that I could select a certain percent of instances and be returned a subset of all the data containing that many instances.

I plotted the genes that appear in the RF on the attached figure. As can be seen in this figure some genes appear much more frequently in the RF than others. This suggests that these features are more important in determining the class of a patient. Since each split in a tree is based on which feature decreases entropy by the most, the genes that appear with higher frequency in the RF are more important/descriptive in separating the classes (AD/non AD patients).

From my data I extracted the following 100 genes that appeared with the highest frequency in the constructed RF. In order to get this data I summed the frequency over all tests (different percentages of feature and instance selection) and determined the genes that had the highest frequency over all tests. I figured that summing the number of occurences of these genes over all trials I ran would be a valid thing to do because

Using 50 percent of features and 70 percent of instances I performed 10-fold cross validation on my RF model. The mean score I got was 0.641795665635 which suggests that this model will have a predictive accuracy of ~64 percent outside of the test data set.



**Support Vector Machine**

1. In the linear kernel you are mapping your instances to a feature space that has the dimensionality of the number of features of each instance. In this case each instance (patient) will be mapped to an 1800 dimensional space (1800 genes) and the support vector machine will find the hyperplane that linearly separates the two classes with the maximal margin. Linear kernels are useful when you have many more features than you have instances, as is the case here. If we are given a weight vector w the features that are the most important will be those that have the highest corresponding weight in the weight vector.

Using this feature selection method my top 100 features are different than the top 100 features from the RandomForest frequency model. The top 100 genes from the linear support vector machine are below.



1. The main difference between the two quadratic kernels is the +1 term in the inhomogeneous quadratic kernel. This can be thought of as a bias or offset term. If the data is not 0 centered, then our classification boundary will go through 0 with a homogeneous quadratic kernel. If our data is not 0 centered and we want to provide some offset term such that data that is say 0.5 centered will be correctly classified, we can use the inhomogeneous kernel which will account for this fact that the data is not 0 centered and classify with a higher accuracy.

In this case the inhomogeneous polynomial kernel with achieved a 63% accuracy when tested with 10-fold cross validation, which was much better than the 51% accuracy that was achieved using a homogeneous polynomial kernel on the same 10-fold cross validation. When I 0 centered the data using the Z-score function the homogeneous polynomial kernel on 0 centered data achieved a similar accuracy to the non 0 centered inhomogeneous polynomial kernel, an accuracy of around 62% using 10-fold cross validation. From these test cases I would say given that the inhomogeneous quadratic kernel would be the best for feature selection because from 10-fold cross validation this model has the highest expected predictive accuracy out of sample.

1. In order to perform feature selection on a quadratic kernel we would use the same method as we did using the linear kernel, except in this case the outputs would be slightly more difficult to understand. As we know the quadratic kernel transforms our input space into a higher dimensional feature space and then solves the classification problem in this space. Because of this the features are transformed and many of the features are combined. For example, in the case of the inhomogeneous quadratic if we are dealing with a two dimensional input space then our transformed function will be of the form.

Because of this our weight vector in our feature space, given a 2-d input space will be a 6-dimensional weight vector with some of the weights in the weight vector corresponding to a combination of variables. In the case of our AD patients we have 1800 dimensional input space which will be transformed to some higher dimensional space using the kernel, the weight vector that is used to classify these examples in the SVM will be in the feature space. If we wanted to examine feature selection, we should simply look at the weight vector and the corresponding terms in the feature transformation. The feature or combination of features in the weight vector with the highest weight will be the most important feature in linearly classifying these data. This would be the method for getting the importance of the features but since expanding a 1800 dimensional input vector into feature space and determining the meaning of the weight vector in feature space is incredibly difficult this will be infeasible.

The magnitude of the different terms in the weight vector will tell us what features or combination of features are the most important in determining the linear classifier in SVM. Because many of the coordinates in the feature space are combinations of several features in the input space (see ) term above, this weight vector will also tell us what combination of features are important in determining classification. So if the magnitude of the weight vector corresponding to these combination terms is high it will tell us that these two features are correlated and have a substantial impact on classification.