Austin Prince

Data Mining

Homework 2

**Preprocessing**

* **Method**

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. I used the Pandas library in python to select the first 1800 genes. Using the following code, you can read in a csv file and only select the first 1800 lines.

data = pd.read\_csv(‘filepath.csv’, nrows=1800)

I wanted to be able to store the two files, case.gex and ctrl.gex in the same dataFrame because in my random forest I will be constructing a random forest using both of these data sets I used the following line of code to concatenate the two data frames (ctrlFrame and copyFrame) into one frame that will contain all of the AD and non-AD patient information.

frames = [ctrlFrame, copyFrame]

finalFrame = pd.concat(frames)

Pandas also has a useful feature that, once your data is in a DataFrame you can specify a percentage sample of this data that you would like to obtain and it will return to you a new DataFrame containing a randomly selected sample containing x% of your original data. I will be using this in constructing my random forest models. If I want a new DataFrame containing a randomly selected sample containing 70% of the original data I can use following line of code.

finalFrame = finalFrame.sample(frac=0.7)

**Imputation**

One thing that I had to with the given data was to impute missing values. There were many entries in both files that contained a NA, and in order to be able to construct an accurate random forest and support vector machine I had to fill in these missing values. I used a Gaussian distribution, the reason for this being two fold. Assuming a Gaussian distribution is a very common practice in the natural sciences and it is often the rule of thumb to use when the actual distribution of a variable is unknown. In addition to to it is is because in statistics you can assume that 95% of data is within 2 standard deviations of the mean, so if we create use a random value within two standard distribution of the mean to fill in our missing data there is a high probability that this value will lie within the normal distribution of values for a given variable.

* Method

In order to fill in the missing values I decided to assume a Gaussian distribution of variables and then fill in the missing values based on a randomly selected number within that Gaussian distribution. For each gene (feature) with a missing value I calculated the Gaussian distribution of this gene using the existing values and then randomly selected a number within the computed Gaussian distribution for the missing value/values. I used the scipy kit for this which using the following line of code can compute the mean and standard deviation of a list of values in python I used the following line of code.

mu, std = norm.fit(listOfVar)

Following this in order to generate a random number within this distribution I used the following line of code which uses pythons numpy library.

rnum = np.random.normal(mu, std)

Once I filled in the missing values were filled in my data was ready to be used to construct a random forest and support vector machine.

**Random Forest Classifier**

* **Method**

To construct the Random Forest classifier, I used the RandomForestClassifer in the scikitlearn package in Python. The RandomForestClassifer function takes several parameters that determine the specific random forest that will be produced. These include n\_estimators (# of trees in the Random Forest), max\_features (% of features sampled at each split of the tree) and criterion (the criteria on which to split the nodes of the tree InfoGain, SplitInfo). In order to make a RandomForest with 500 trees using 70% of the model you can use the following line of code.

rf = RandomForestClassifier(n\_estimators=500, criterion=’entropy’, max\_features=70)

In order to fit the training data, you can use the following line of code which actually constructs the random forest. In this case X is my input features and y are the corresponding labels.

rf.fit(X, y)

The Random Forest model in python has an attribute called .estimators\_ which is a list containing all of the individual decision trees that are in the Random Forest. In order to get the individual trees I used the following line of code.

estimators = rf.estimators\_

Once I have the individual trees I can parse through these trees one by one and determine which features these trees were split on. In order to do this, I have to use the list of estimators (individual decision trees in the random forest) and parse through each tree and extract the features that were split on. If I call the following code ,I will get a list of features that the tree was split on.

features = estimator.tree\_.feature

Once I have the list of features that each tree in the random forest split on I can add up the occurrence of each individual feature over all trees in the random forest and determine which features are split on the most often.

**Random Forest Discussion**

If we want to use the random forest model in order to determine which features are the most important it is instructive to look at the features that are split on most often. Each decision tree in the random forest splits on 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 decision trees by randomly selecting certain features. If a gene (feature) is split on more often this means that this gene will be more descriptive in discriminating between the two classes (AD 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.

One factor to consider is that the root node of a decision tree is the most important node in determining which class a sample will fall under. The attribute that this root node splits on will be the most important split in the data, and will thus be the most descriptive in describing the split between the classes. Because I doing 18 trials in which I construct a random forest containing 500 trees I figured that given the randomness of the features included the data set to construct of each these trees over all of these trials the gene that appears the most often would be the most important.

In order to compare two genes based on their frequency we can say that the gene that is more frequent in the random forest will be the more important feature in determining whether a patient has AD or not. This is based on the above logic that if a feature is split on, then it is better in discriminating between the two classes and so if this feature is split on with a higher frequency in a random forest than another then it must reduce the entropy of the dataset more than the other feature, thus it splits the dataset into classes better than the other feature.

For this homework I ran 18 trials of random forests, each containing 500 trees. First I fixed the number of features (genes) at 70% of the total number of features, and varied the number of instances from 50% to 90%, with an increment of 5% and then I fixed the number of instances at 70% of the total number of instances, and varied the number of features (genes) from 50% to 90%, with an increment of 5%. Once I had the data on the gene frequency from these 18 trials I average the frequency of each gene over all 18 of the trials to determine the 100 most important genes. I figured that averaging the frequency over the 18 trials was valid because it allows us to see which genes are the most important over a range of features and instances. Averaging over a range of features is valid because it introduces randomness that allows us to see which genes overall are the most important. Averaging over a range of instances (patients) is valid because it allows us to see the trends of which genes are the most important overall and gets rid of patient to patient variability of genes that cause AD.

Below I have included a list of the 100 most important genes and a graph with the occurrences of the 100 most important genes over the 9000 trees that were constructed in 18 random forests.



One last bit of information I want to include about the random forest model is that after running 10-fold cross validation using 50 percent of features and 70 percent of instances I got a mean score of 0.641795665635 which suggests that this model will have a predictive accuracy of ~64 percent outside of the test data set.

**Support Vector Machine**

* **Method**

To construct a support vector machine classifier, I used pythons scikitlearn SVC (support vector classifier) package. The SVC package takes several parameters which specify the particular classifier that will be produced. These include kernel (specifies the type of kernel to use options include, linear, polynomial, radial basis function…etc), degree (determines the degree of the polynomial kernel) and coef0 (determines the homogeneity of the polynomial kernel if coef0=1 then we have an inhomogeneous polynomial kernel similar to the one that is presented in class). Below I will supply the code that can be used to generate a support vector machine with a linear kernel, a homogeneous polynomial kernel and an inhomogeneous polynomial kernel.

Linear kernel

clf = svm.SVC(kernel=’linear’)

Homogeneous quadratic kernel

clf = svm.SVC(kernel=’poly’, degree=2)

Inhomogeneous quadratic kernel

clf = svm.SVC(kernel=’poly’, degree=2, coef0=1)

In order to fit the training data, you can use the following line of code which actually constructs the support vector classifier. In this case X is my input features and y are the corresponding labels.

clf.fit(X, y)

In order to get the coefficients of the weight vector, which is only available for the linear kernel you can access the .coef\_ property of the support vector classifier, which can be done using the following code.

coefficients = clf.coef\_

**Support Vector Discussion**

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. Weights with a value of 0 means that the corresponding feature will not contribute to the classifying hyperplane. When classifying an example, the outputted value will be a linear combination of the features with the weights in the weight vector as coefficients, therefore the features with a higher weight in the weight vector will be more decisive in classifying your data.

Using this feature selection method my top 100 features have some overlap with the the top 100 features from the random forest frequency model. The top 100 genes from the linear support vector machine are below. In order to ensure consistency of my model I ran the support vector machine 100 times and although the top 100 features were in a different order every time they were consistent over these 100 trials.



In comparing these genes to the genes that I found using the random forest frequency model I found that out of all of the 100 top genes in both cases there were 12 genes that were the same in both lists. This is perhaps due to the classification error of both of these models, in that both models have their own biases in fitting the test data and that is coming out in the fact that the important features of each model is different.

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. The 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 not 0 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 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. This means that the data is not 0 centered and that the feature vectors are shifted in some way. 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. When the data was normalized it was also revealed that both the homogeneous and inhomogeneous kernels performed equally well (~62% accuracy using 10-fold cross validation). This is because when we normalize the data the bias term of the inhomogeneous kernel will have minimal effect on our model. From these test cases I would say given 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 kernel if we are dealing with a two dimensional input space then our transformed input vector will be translated to the following form in feature space.

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 features. 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 component of the vector in features space (ie or any others from above) . 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 an 1800 dimensional input vector into feature space and determining the meaning of the weight vector in the original input space is incredibly difficult (much more than in the above example with a 2-d input space) this will be essentially infeasible.

The magnitude of the different terms of the weight vector in the feature space 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 or more features are correlated and have a substantial impact on classification. In the case of our 1800 dimensional input data because the weight vector is classifying in feature space finding the transformation and what each term means in input space is very difficult and will make this for all purposes impossible.