gene

November 22, 2015

1 ECS 171 Assignment 3

1.1 Zhen Zhang (last four digit of ID: 5403)

1.2 Nov 22, 2015

First of all, I shall say that before doing the main job in python, I use R to process the data. The script is listed below, with doing these jobs one by one:

```
# read the data, and use regular expression to extract some identifiers
gene <- scan("ecs171.dataset.txt", what = ’’)
pattern <- grep("T[0-9]{3}", gene)
gene_index_start <- c(1, pattern)
gene_index_end <- c(pattern - 1, length(gene))
row_num <- length(gene_index_start)
col_num <- gene_index_end[1] - gene_index_start[1] + 1

# generate the raw data
gene_data_raw <- Map(function(i, j) gene[i:j], gene_index_start, gene_index_end)

colname <- gene_data_raw[[1]]

# if the data is shorter than the required length, add 0 at last
invisible(lapply(1:row_num, function(i) {
  if (length(gene_data_raw[[i]]) != col_num) gene_data_raw[[i]] <<- c(gene_data_raw[[i]], 0)
}))

# combine them into a dataframe
gene_data <- data.frame(do.call(’rbind’, gene_data_raw), stringsAsFactors = FALSE)

# drop the first row, which is the name
gene_data <- gene_data[-1, ]

# add column name
colnames(gene_data) <- colname

# drop the last column
gene_data <- gene_data[, -length(gene_data)]

# generate the combination of Medium and Stress data, which will be used in Question 5.
Medium_Stress <- unlist(Map(paste, gene_data$Medium, gene_data$Stress, sep = ’.’))
gene_data[, 1] <- Medium_Stress
names(gene_data)[1] <- ’Medium_Stress’
gene_data$Medium_Stress <- as.numeric(as.factor(gene_data$Medium_Stress))
```

```
# transform relevant data to double type
invisible(lapply(2:5, function(i) {
  gene_data[[i]] <<- as.numeric(as.factor(gene_data[[i]]))
}))

# generate the final csv file
write.csv(gene_data, "~/Desktop/gene_data.csv", row.names = F)
```

Now let's begin this homework, and all the following codes are written in python.

```
In [1]: # import the needed modules
        import numpy as np
        import pandas as pd
        import warnings
        warnings.filterwarnings('ignore')
```

```
In [2]: # read into the data
        gene = pd.read_csv("/Users/Elliot/Desktop/gene_data.csv", header = 0)
        gene.data = gene.iloc[:, 6:]
```

I need to split the data, and following python's convention, I will give the splitted data some properties, including data, growthRate and so on.

1.3 Question 1

```
In [3]: # import the needed modules used in this question
        from matplotlib import pyplot as plt
        from sklearn.linear_model import Lasso, LassoCV
        from sklearn.cross_validation import KFold
```

Generate the cross validation folds:

```
In [4]: # cross-validation preparation
        kf = KFold(gene.data.shape[0], n_folds=10)
```

Fit the model

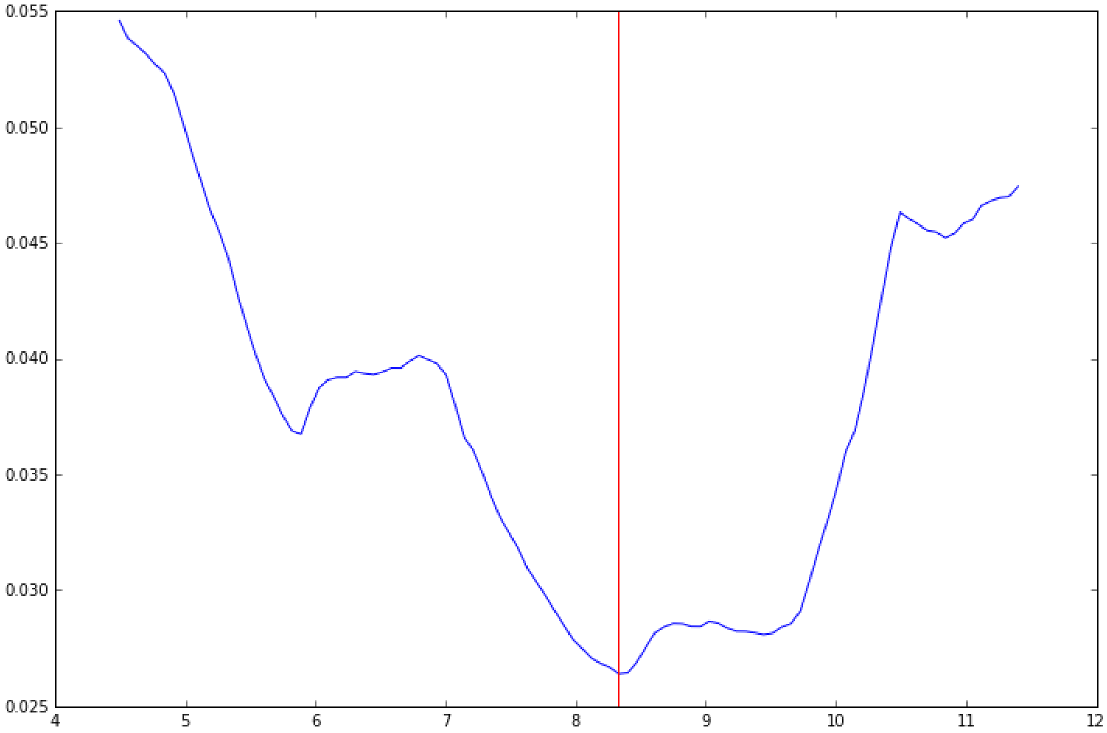
```
In [5]: # fit
        lasso_model = LassoCV(cv=10, copy_X=True, normalize=True)
        lasso_fit = lasso_model.fit(gene.data, gene.GrowthRate)
```

Let's look at a plot to see the general tendency of the result

```
In [6]: %matplotlib inline
        plt.rcParams['figure.figsize'] = (12, 8)
```

```
In [7]: # Note here I made a negative log transformation to express the plot easier to see and interpre
        plt.plot(-np.log(lasso_fit.alphas_), lasso_fit.mse_path_.mean(axis=1))
        plt.axvline(-np.log(lasso_fit.alpha_), color='red')
```

```
Out[7]: <matplotlib.lines.Line2D at 0x1131f4630>
```



See the value of optimal constrained parameter value

```
In [8]: print('The optimal optimal constrained parameter value is: ', lasso_fit.alpha_)
```

```
The optimal optimal constrained parameter value is:  0.000241053628825
```

Now let's see which coefficient is not equal to zero:

```
In [9]: lasso_coefs_index = np.where(lasso_fit.coef_ != 0)
        lasso_coefs_name = gene.data.columns[lasso_coefs_index].tolist()
        print('The features that have non-zero coefficients are:')
        print(lasso_coefs_name)
```

```
The features that have non-zero coefficients are:
['b2353', 'b1890', 'b4131', 'b1452', 'b2505', 'b1262', 'b1463', 'b3688', 'b1673', 'b0326', 'b0440', 'b35
```

1.4 Question 2

Here I will talk about my bootstrap algorithm.

Algorithm:

First of all, I will do the bootstrap `bs_num` times. In each loop, I select the number of samples equal to `n_samples` (the number of the original sample), and based on these selecte indices (`bs_index`) and the lasso regularized features (`lasso_coefs_index`) from the previous question, I can construct the bootstrap sample (`X_bs`). Then use lasso method to predict the output of the whole original data, then I can get the prediction (`lasso_bs_pred`). Over all loops, I can get a matrix of dimension `n_samples` * `bs_num`.

Now, I can first sort all the results each row (each sample), then select the one ranks 2.5% and 97.5% in each sample, which will be the bootstrap lower value and upper value for the 95% confidence interval.

Assumption:

Each sample is a valid representation of the population.

Each sample is identical and independent distribution (i.i.d).

```
In [10]: # bs is short for bootstrap
         bs_num = 100
         n_samples = gene.GrowthRate.shape[0]
         gene_bs_pred = np.empty((n_samples, bs_num))
         gene_features_lasso_data = gene.data[gene.data.columns[lasso_coefs_index]]

         for i in range(bs_num):
             bs_index = np.random.choice(n_samples, n_samples, replace=True)
             X_bs = gene_features_lasso_data.iloc[bs_index, :]
             y_bs = gene.GrowthRate[bs_index]
             lasso_bs_pred = lasso_model.fit(X_bs, y_bs).predict(gene_features_lasso_data)
             gene_bs_pred[:, i] = lasso_bs_pred

         gene_bs_pred_sort = np.sort(gene_bs_pred)
         gene_bs_pred_result = gene_bs_pred_sort[:, [3, 97]]

         print(gene_bs_pred_result)
```

```
[[ 0.60079666  0.6378432 ]
 [ 0.61049767  0.65339268]
 [ 0.60645946  0.64422516]
 [ 0.70600731  0.77084599]
 [ 0.67327522  0.70573273]
 [ 0.68988926  0.72669264]
 [ 0.43178885  0.48330707]
 [ 0.46152625  0.49407513]
 [ 0.4672918   0.49847266]
 [ 0.4683631   0.4988734 ]
 [ 0.47314322  0.5045316 ]
 [ 0.46680257  0.50248761]
 [ 0.43345448  0.489958  ]
 [ 0.43345448  0.489958  ]
 [ 0.43099659  0.50271311]
 [ 0.45623726  0.50014108]
 [ 0.41428568  0.48190931]
 [ 0.44246028  0.49537041]
 [ 0.48353099  0.5340476 ]
 [ 0.46639823  0.53589208]
 [ 0.48705988  0.76859314]
 [ 0.3217138   0.6499975 ]
 [ 0.47333042  0.52820138]
 [ 0.34620826  0.6480973 ]
 [ 0.37202753  0.40721005]
 [ 0.36825416  0.41364717]
 [ 0.37683095  0.42543649]
 [ 0.33533449  0.44074919]
 [ 0.4068336   0.44579194]
 [ 0.37118433  0.464534  ]
 [ 0.44963209  0.47766856]
 [ 0.45730868  0.4891397 ]
 [ 0.44688932  0.50135854]
 [ 0.45828957  0.50426907]
 [ 0.47377107  0.51325002]
 [ 0.46482616  0.50376451]
 [ 0.62078094  0.65571583]
 [ 0.60706669  0.64098726]
 [ 0.63230004  0.67266641]
 [ 0.59384872  0.64141587]
 [ 0.60623839  0.64550507]
 [ 0.56980925  0.64669662]
 [ 0.62224793  0.68182475]
 [ 0.37045069  0.97074242]
 [ 0.67726766  0.7268884 ]
 [ 0.68395986  0.72103193]
 [ 0.67945355  0.72426457]
 [ 0.24123736  0.30161473]
 [ 0.25335944  0.33959153]
 [ 0.2950368   0.34324766]
 [ 0.29357728  0.36956827]
 [ 0.40061841  0.47589585]
 [ 0.41326564  0.4986225 ]
 [ 0.41170727  0.47808561]
 [ 0.29124673  0.52254403]
 [ 0.10213294  0.19365921]
 [ 0.1243096   0.20368538]
 [ 0.18531798  0.26908826]
 [ 0.3533788   0.40398342]
 [ 0.20045925  0.24168558]
 [ 0.21570762  0.25884199]
 [ 0.20938494  0.25185316]
 [ 0.15735748  0.31790334]
 [ 0.12485299  0.2529779 ]
 [ 0.15138975  0.25888247]
 [ 0.11001208  0.2843101 ]
 [ 0.34528868  0.45420656]
 [ 0.32995863  0.40338915]
 [ 0.37107444  0.43844274]
 [ 0.3320154   0.39659607]
 [ 0.33861895  0.39308612]
 [ 0.34049281  0.38814181]
 [ 0.352577    0.40910445]
 [ 0.3332708   0.40457743]
 [ 0.3628511   0.41027048]
 [ 0.17314462  0.28593983]
 [ 0.255175    0.32542621]
 [ 0.23119773  0.29187865]
 [ 0.19013608  0.30521651]
 [ 0.23863126  0.31725473]
 [ 0.21401301  0.30266643]
 [ 0.0975463   0.16583652]
 [ 0.1012194   0.19899578]
 [ 0.10455617  0.23280175]
 [ 0.0539116   0.15994388]
 [ 0.11132889  0.17897694]
 [ 0.0433974   0.68952171]
 [ 0.06607152  0.12182055]
 [ 0.08253911  0.1324048 ]
 [ 0.05540445  0.11716323]
 [ 0.08861284  0.14444651]
 [ 0.02199712  0.09727461]
 [ 0.04662368  0.11030945]
 [-0.00920061  0.07172699]
 [ 1.19902223  1.28997628]
 [ 1.16795976  1.28423665]
 [ 1.11112729  1.50642585]
 [ 1.15479744  1.32470909]
 [ 1.22273649  1.2927238 ]
 [ 0.58574416  0.68379836]
 [ 0.59777406  0.65335592]
 [ 0.59582104  0.66235727]
 [ 0.64247333  0.70238423]
 [ 0.56212779  0.70461669]
 [ 0.63409115  0.71161482]
 [ 0.15314746  0.47793652]
 [ 0.1701859   0.48877465]
 [ 0.27876299  0.65601188]
 [ 0.00874264  0.17769519]
 [ 0.10732558  0.2770593 ]
 [ 0.14371058  0.27032204]
 [ 0.11163261  0.33985501]
 [ 0.0519918   0.41721732]
 [ 0.16874892  0.38670247]
 [ 0.08661757  0.17709105]
 [ 0.0732238   0.31737588]
 [ 0.14301807  0.25420155]
 [ 0.09515459  0.14824428]
 [ 0.09981855  0.15029836]
 [ 0.07001355  0.1557102 ]
 [ 0.10338175  0.16067499]
 [ 0.10232789  0.16142176]
 [ 0.04462867  0.1286822 ]
 [ 0.08072236  0.1445466 ]
 [ 0.09788782  0.15126642]
 [ 0.10166659  0.16622541]
 [ 0.09076169  0.14444214]
 [ 0.10044322  0.16552702]
 [ 0.08508728  0.19289526]
 [ 0.45252317  0.80199114]
 [ 0.4715215   0.71129048]
 [ 0.30029481  0.69665409]
 [ 0.52714106  0.77687741]
 [ 0.51293747  0.87156862]
 [ 0.48304953  0.73995966]
 [ 0.40351005  0.59873651]
 [ 0.34929786  0.65172234]
 [ 0.49560546  0.76913504]
 [ 0.53619432  0.72439752]
 [ 0.18419483  0.25248869]
 [ 0.5509977   0.6277938 ]
 [ 0.50967126  0.83099283]
 [ 0.39300361  0.43311305]
 [ 0.38975792  0.42060333]
```

```
 [ 0.37681293  0.4132786 ]
 [ 0.38220732  0.41954822]
 [ 0.3996164   0.41923537]
 [ 0.37127303  0.4164622 ]
 [ 0.36905695  0.40843571]
 [ 0.36485545  0.39674961]
 [ 0.3936861   0.41604959]
 [ 0.37467028  0.40754652]
 [ 0.39135767  0.4136407 ]
 [ 0.38712747  0.44045721]
 [ 0.3746371   0.40411885]
 [ 0.3673239   0.38801153]
 [ 0.4058544   0.48217256]
 [ 0.1937552   0.27380398]
 [ 0.5846476   0.70315035]
 [ 0.53301632  0.60693865]
 [ 0.53592979  0.60496527]
 [ 0.24196132  0.30818149]
 [ 0.26274309  0.33980131]
 [ 0.26982101  0.35479431]
 [ 0.21213672  0.29854755]
 [ 0.16575757  0.23548758]
 [ 0.22050629  0.33438882]
 [ 0.11101277  0.43373043]
 [ 0.29734736  0.45017653]
 [ 0.28842978  0.3489017 ]
 [ 0.20121946  0.36865559]
 [ 0.51571257  0.61204554]
 [ 0.20661756  0.37504634]
 [ 0.26778795  0.48341264]
 [ 0.47198364  0.60112985]
 [ 0.46002591  0.69374054]
 [ 0.42766007  0.60365341]
 [ 0.46756224  0.54924289]
 [ 0.42860072  0.51692897]
 [ 0.19051409  0.23136264]
 [ 0.18694794  0.21828075]
 [ 0.19995562  0.22442595]
 [ 0.18989785  0.22384015]
 [ 0.17806147  0.20700236]
 [ 0.1947593   0.23771555]
 [ 0.18552646  0.22662168]
 [ 0.17386629  0.21837001]
 [ 0.19807735  0.22312054]
 [ 0.18228541  0.20448211]
 [ 0.16931399  0.19960045]
 [ 0.18755388  0.22599847]
 [ 0.18010133  0.21705941]
 [ 0.19204738  0.2196252 ]
 [ 0.18823389  0.23088988]]
```

1.5 Question 3

First I need to get the mean value of all genes:

```
In [11]: gene_features_lasso_data_mean = gene_features_lasso_data.apply(np.mean, axis = 0)
```

Use the lasso method to predict the growth rate for the mean gene:

```
In [12]: gene_features_lasso_data_mean_predict_value = lasso_fit.predict(gene_features_lasso_data_mean.
```

```
In [13]: print('The predicted growth rate for a bacterium whose genes are expressed exactly at the mean
         print(gene_features_lasso_data_mean_predict_value)
```

```
The predicted growth rate for a bacterium whose genes are expressed exactly at the mean expression valu
[ 0.38668247]
```

1.6 Question 4

First I need to talk about what I am doing afterwards, since all question 4 to question 6 are all using the same functions to accomplish the task.

The first step is to import the libraries related to this part. It includes SVM, AUC, AUPRC, Cross-Validation and so on.

The second step is to define several functions which will be used afterwards. They are:

`feature_selection`: It will select the feature. Unlike the linear lasso method used in question 1, now I use a lasso method based on SVM of linear kernel. Since they are all L1 norm, they all have the ability of feature selection.

`svm_cv_score`: This function can calculate the svm score, which will be rendered to calculate the AUC and AUPRC. It accepts the data (X), target (y), and the train-test split index, and give a result of svm score.

`to_binary`: This function translates y to binary values. For example, if y has 6 classes, then translated binary y will be a 6 dimension vector, such as (0, 0, 1, 0, 0, 0).

`compute_roc`: This function calculates the FPR (False Positive Rate), TPR (True Positive Rate) and ROC Curve values. It accepts the y value and y score of SVM classification, then calculate the results. Note here to generate a overall performance for all the classifiers, I create a micro-average curve, and plot it individually.

`compute_pr`: This function follows the same logic as function `compute_roc`, but is calculating Recall, Precision and AUPRC Curve values.

The third step is to combine them together and plot the ROC and AUPRC curve, and this is what the function `svm_cv_roc` is doing. It accepts data (X) and target (y) as input, first it will select the feature based on L1 norm, and make the y value to binaries. Then generate the 10-fold cross-validation train-validation split, and initialize the y score matrix. After that, the y score is calculated, and use it to calculate AUC and AUPRC. Then plot them.

The fourth step is to see the plots. In the plots, I have list all the classification curves with respect to different classes and a micro-average curve for an overall performace evaluation. I also give the area value under each curve to facilitate comparison.

There are also some comments below. So if you still have some question, please see these comments.

```
In [14]: # import svm library and other required functions
         from sklearn.svm import LinearSVC
         from sklearn.feature_selection import SelectFromModel
         import matplotlib.pyplot as plt
         from sklearn import svm, datasets
         from sklearn.metrics import roc_curve, auc
         from sklearn.metrics import precision_recall_curve
         from sklearn.metrics import average_precision_score
         from sklearn.cross_validation import train_test_split
         from sklearn.preprocessing import label_binarize
         from sklearn.multiclass import OneVsRestClassifier
         from scipy import interp
         from sklearn.svm import SVC
```

```
In [15]: def feature_selection(X, y):
             # This function is the feature selection using the method of L1, which is equivalent to la
             # but is done inside LinearSVC.

             # set L1 penalty and select
             lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)
             model = SelectFromModel(lsvc, prefit=True)
             # get reduced X
             X_new = model.transform(X)
             # get the number of feature selected
             n_feature = X_new.shape[1]

             return X_new, n_feature


         def svm_cv_score(X, y, svc, train, test):
             # This function does the cross validation for each train and test combination in svm

             # get the train and test data based on the index
             X_train, X_test = X[train, :], X[test, :]
             y_train, y_test = y[train], y[test]
             # fit the data, get svm score to plot roc
             y_score = svc.fit(X_train, y_train).decision_function(X_test)

             return y_score


         def to_binary(y):
             # This function converts a categorical vector into a binary format

             n_classes = np.max(y)
             y = label_binarize(y, classes=range(n_classes))

             return y


         def compute_roc(y, y_score, n_classes):
             # This function accepts y value and the corresponding lasso score, then calculate the
             # fpr, tpr, AUC.

             fpr = dict()
             tpr = dict()
             roc_auc = dict()
             for i in range(n_classes):
                 fpr[i], tpr[i], _ = roc_curve(y[:, i], y_score[:, i])
                 roc_auc[i] = auc(fpr[i], tpr[i])

             # Compute micro-average ROC curve and ROC area
             fpr["micro"], tpr["micro"], _ = roc_curve(y.ravel(), y_score.ravel())
             roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

             return fpr, tpr, roc_auc
```

```
        def compute_pr(y, y_score, n_classes):
            # This function accepts y value and the corresponding lasso score, then calculate the
            # precision, recall, AUPRC

            precision = dict()
            recall = dict()
            average_precision = dict()
            for i in range(n_classes):
                precision[i], recall[i], _ = precision_recall_curve(y[:, i], y_score[:, i])
                average_precision[i] = average_precision_score(y[:, i], y_score[:, i])

            # Compute micro-average PR cirve and AUPRC area
            precision["micro"], recall["micro"], _ = precision_recall_curve(y.ravel(), y_score.ravel()
            average_precision["micro"] = average_precision_score(y, y_score, average="micro")

            return precision, recall, average_precision
```

```
In [16]: def svm_cv_roc(X, y):
            # This function is the main function that connects different jobs of doing svm cross valid

            # Preparation
            n_sample = X.shape[0]
            n_classes = np.max(y)

            # Select which feature should be included in the model
            X, n_feature = feature_selection(X, y)

            print("The number of features selected for this variable is ", n_feature, '\n')

            # Make the output Binary
            y = to_binary(y)

            # split the data into cross-validation sets
            kf = KFold(X.shape[0], n_folds=10, shuffle=True)
            svc = OneVsRestClassifier(SVC(kernel='linear', probability=True))
            y_score = np.empty((n_sample, n_classes))

            # Do cross-validation
            for _, (train, test) in enumerate(kf):
                y_score_temp = svm_cv_score(X, y, svc, train, test)
                y_score[test] = y_score_temp

            # Compute ROC curve and ROC area for each class
            fpr, tpr, roc_auc = compute_roc(y, y_score, n_classes)

            # Compute Precision-Recall and plot curve
            precision, recall, average_precision = compute_pr(y, y_score, n_classes)

            # Next part are all plot functions

            # Plot micro ROC curve
            plt.figure()
            plt.plot(fpr["micro"], tpr["micro"], label='micro-average ROC curve (area = {0:0.2f})'
                                                       ''.format(roc_auc["micro"]), linewidth=2)
```

```
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('The overall ROC curve Performance')
plt.legend(loc="lower right")
plt.show()

# Plot FPR-TPR curve for each class
plt.figure()
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (area = {1:0.2f})'
                                   ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('The ROC curve for each class')
plt.legend(loc="lower right")
plt.show()

# Plot micro PR curve
plt.clf()
plt.plot(recall["micro"], precision["micro"],
         label='micro-average Precision-recall curve (area = {0:0.2f})'
               ''.format(average_precision["micro"]))

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Extension of Precision-Recall curve to multi-class')
plt.legend(loc="lower right")
plt.show()

# Plot Precision-Recall curve for each class
for i in range(n_classes):
    plt.plot(recall[i], precision[i],
             label='Precision-recall curve of class {0} (area = {1:0.2f})'
                   ''.format(i, average_precision[i]))

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Extension of Precision-Recall curve to multi-class')
plt.legend(loc="lower right")
plt.show()
```

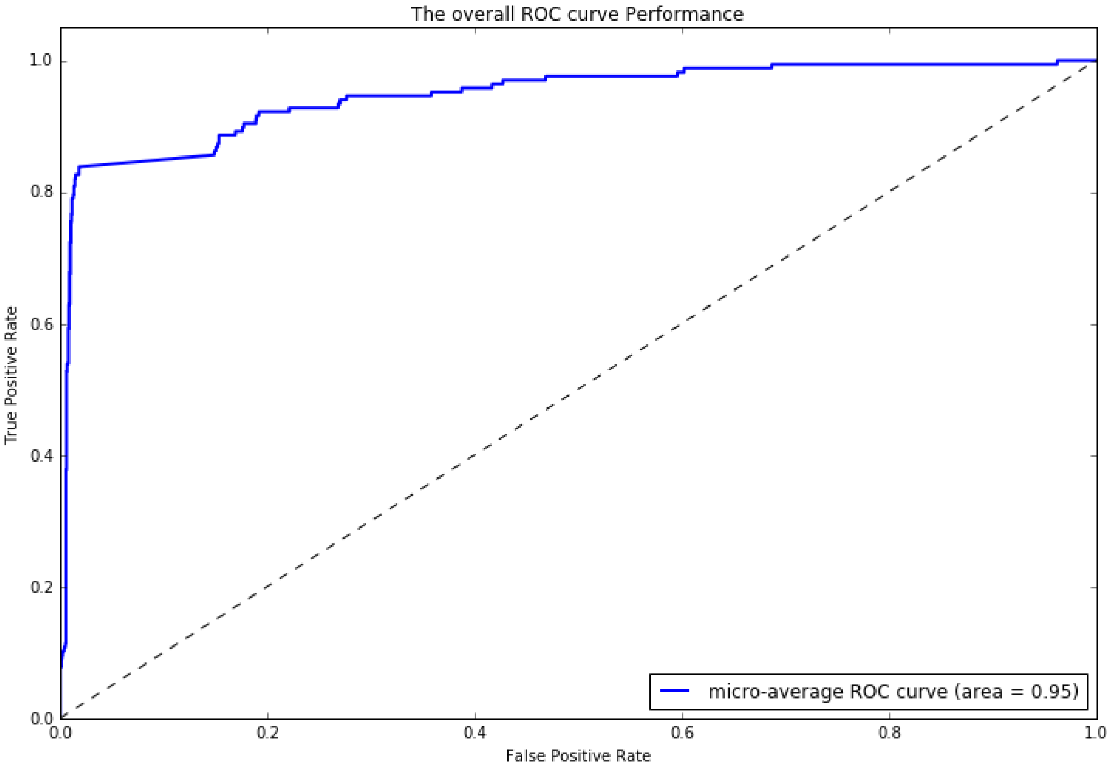
```
            return
```

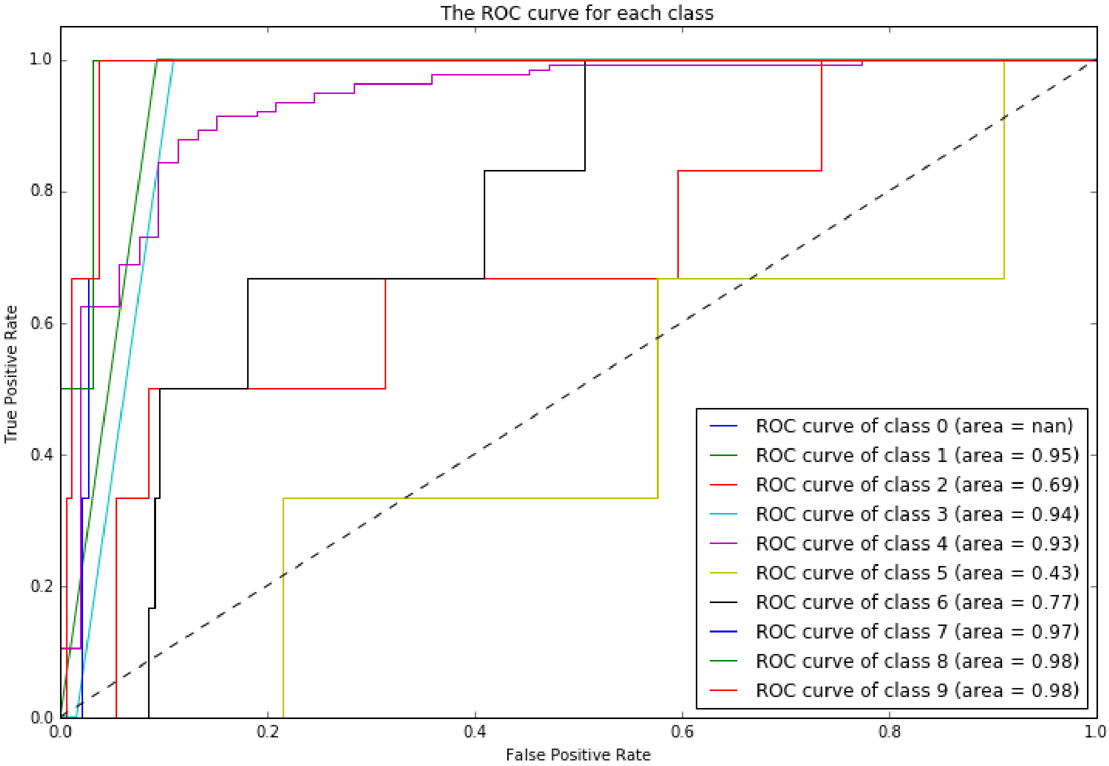
Let's see each classifier related to each variable one by one.

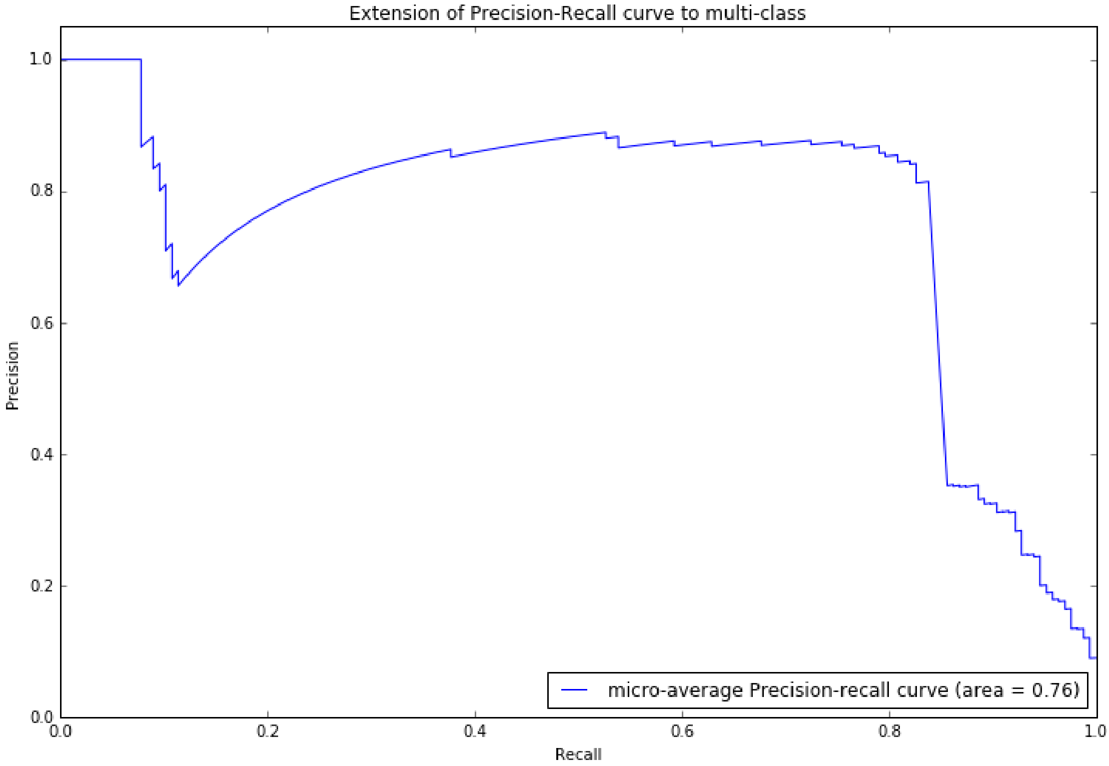
Strain:

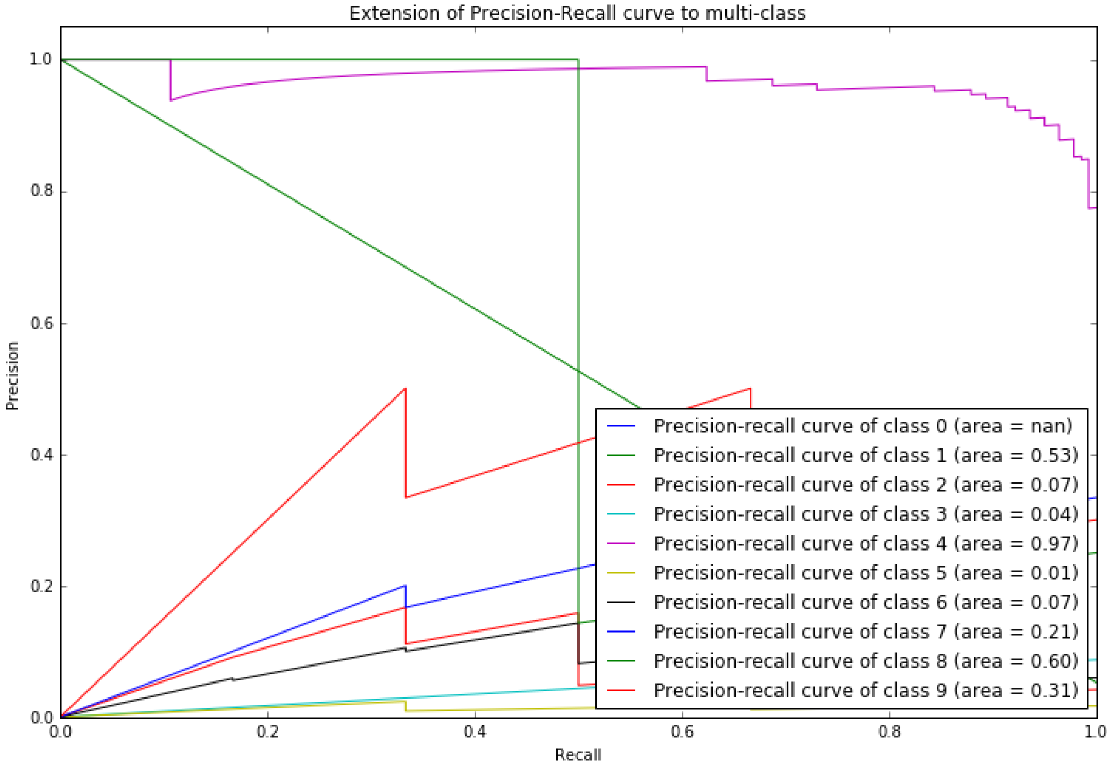
```
In [17]: svm_cv_roc(gene.data, gene.Strain)
```

```
The number of features selected for this variable is  18
```




The ROC curve for each class
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = nan)
ROC curve of class 1 (area = 0.95)
ROC curve of class 2 (area = 0.69)
ROC curve of class 3 (area = 0.94)
ROC curve of class 4 (area = 0.93)
ROC curve of class 5 (area = 0.43)
ROC curve of class 6 (area = 0.77)
ROC curve of class 7 (area = 0.97)
ROC curve of class 8 (area = 0.98)
ROC curve of class 9 (area = 0.98)

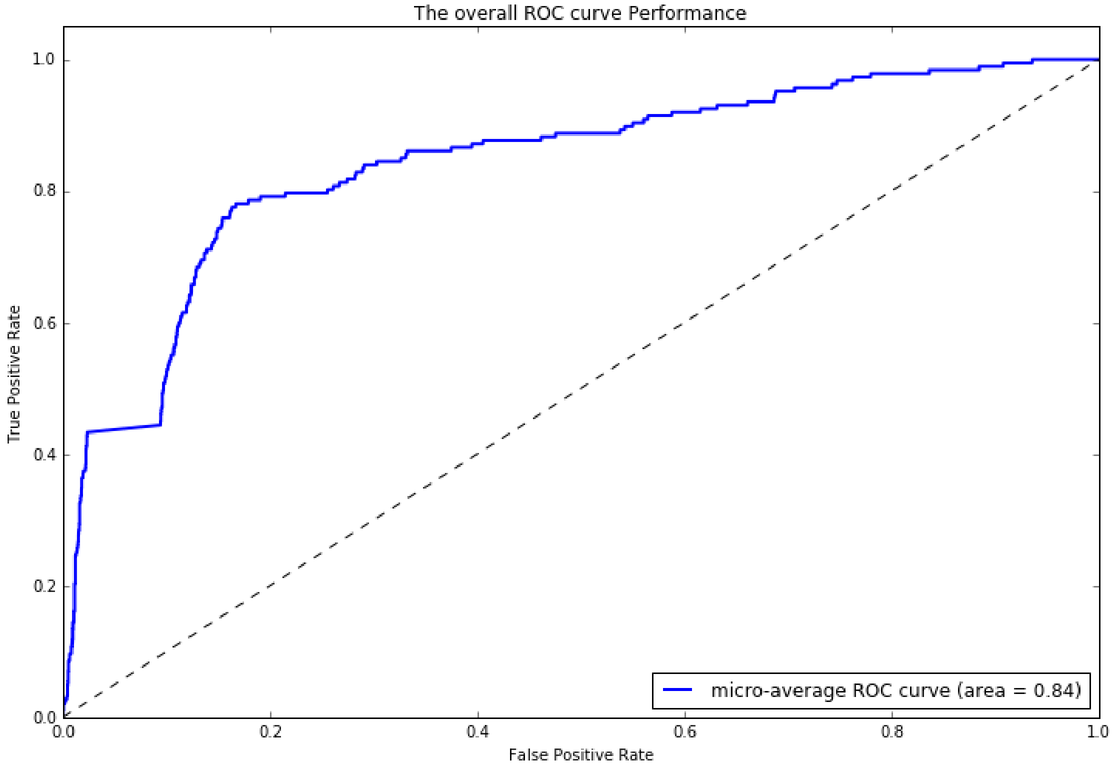

Extension of Precision-Recall curve to multi-class
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
micro-average Precision-recall curve (area = 0.76)

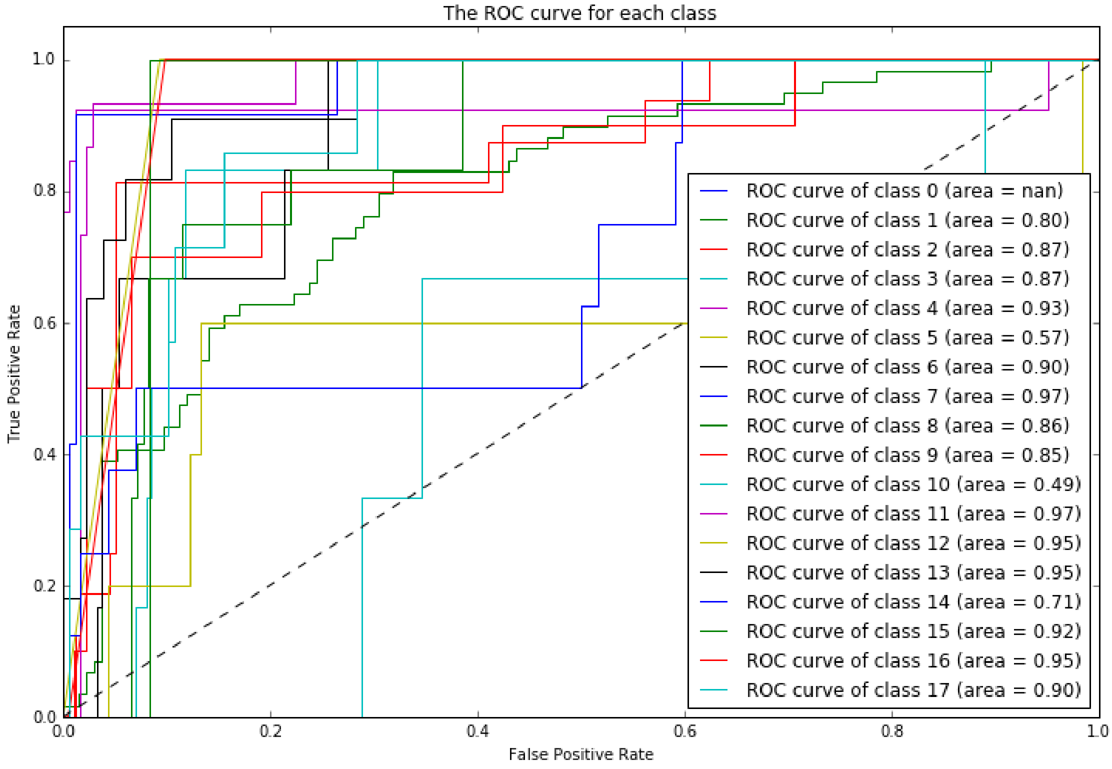


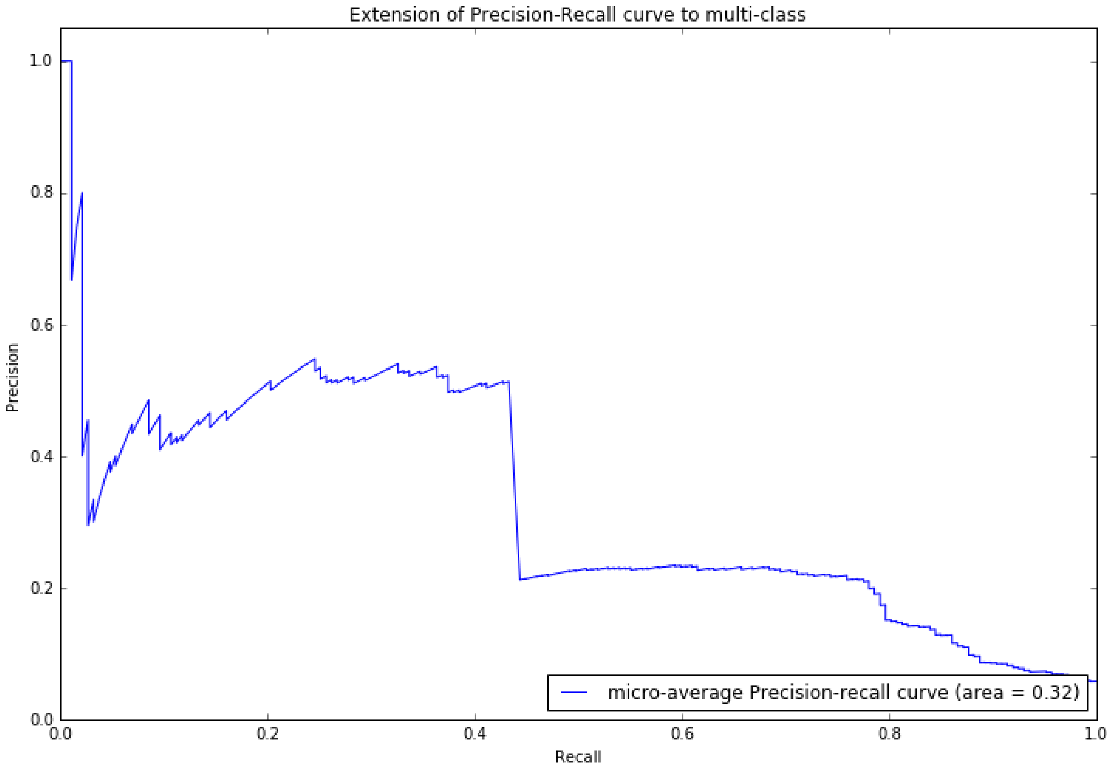
Medium:

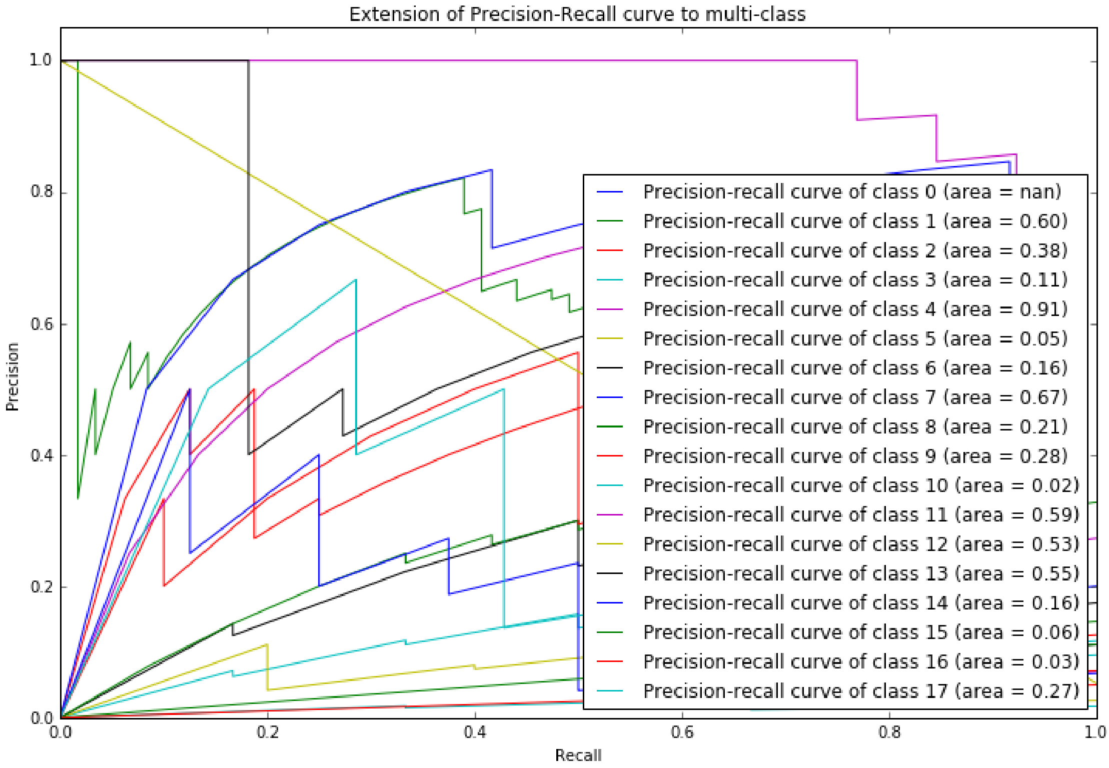
```
In [18]: svm_cv_roc(gene.data, gene.Medium)
```

```
The number of features selected for this variable is  29
```


The overall ROC curve Performance
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
micro-average ROC curve (area = 0.84)


The ROC curve for each class
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = nan)
ROC curve of class 1 (area = 0.80)
ROC curve of class 2 (area = 0.87)
ROC curve of class 3 (area = 0.87)
ROC curve of class 4 (area = 0.93)
ROC curve of class 5 (area = 0.57)
ROC curve of class 6 (area = 0.90)
ROC curve of class 7 (area = 0.97)
ROC curve of class 8 (area = 0.86)
ROC curve of class 9 (area = 0.85)
ROC curve of class 10 (area = 0.49)
ROC curve of class 11 (area = 0.97)
ROC curve of class 12 (area = 0.95)
ROC curve of class 13 (area = 0.95)
ROC curve of class 14 (area = 0.71)
ROC curve of class 15 (area = 0.92)
ROC curve of class 16 (area = 0.95)
ROC curve of class 17 (area = 0.90)

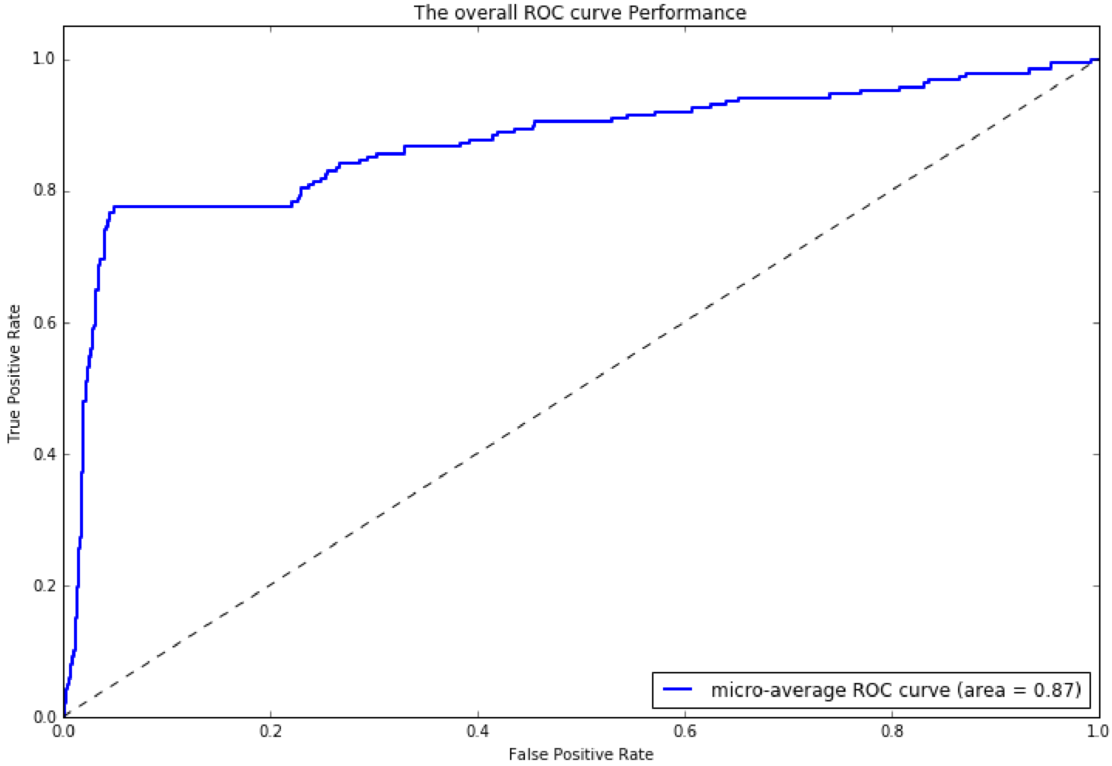

Extension of Precision-Recall curve to multi-class
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
micro-average Precision-recall curve (area = 0.32)

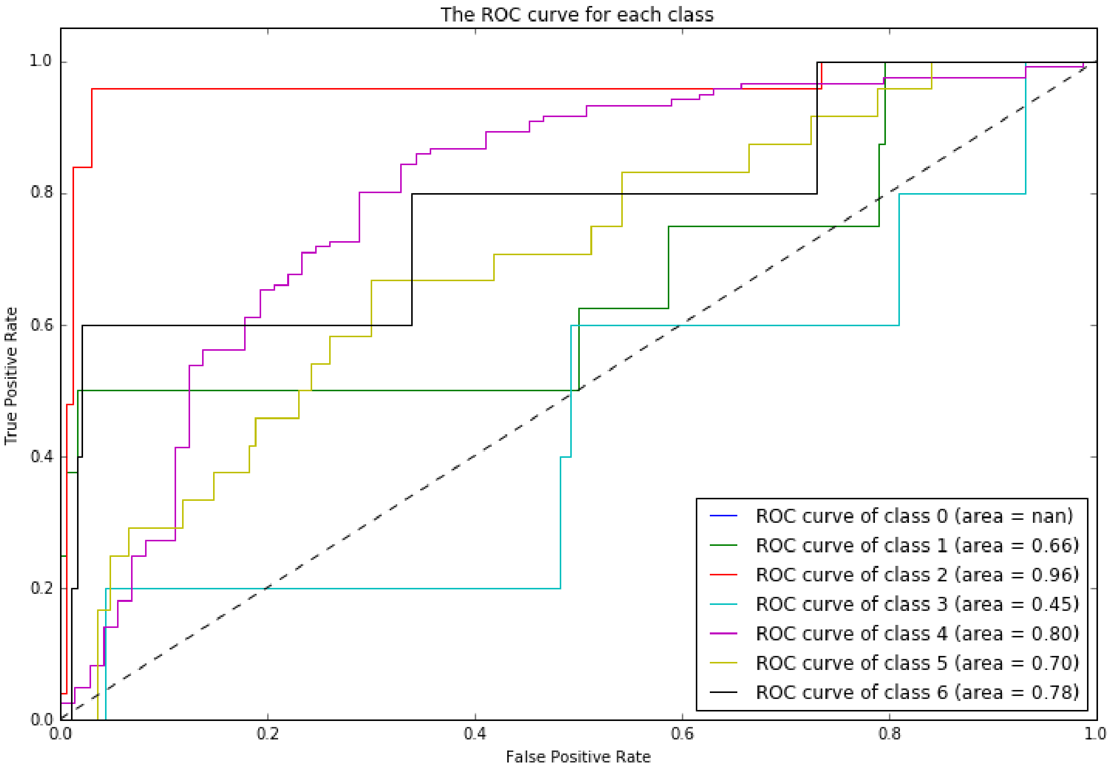

Extension of Precision-Recall curve to multi-class
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Precision-recall curve of class 0 (area = nan)
Precision-recall curve of class 1 (area = 0.60)
Precision-recall curve of class 2 (area = 0.38)
Precision-recall curve of class 3 (area = 0.11)
Precision-recall curve of class 4 (area = 0.91)
Precision-recall curve of class 5 (area = 0.05)
Precision-recall curve of class 6 (area = 0.16)
Precision-recall curve of class 7 (area = 0.67)
Precision-recall curve of class 8 (area = 0.21)
Precision-recall curve of class 9 (area = 0.28)
Precision-recall curve of class 10 (area = 0.02)
Precision-recall curve of class 11 (area = 0.59)
Precision-recall curve of class 12 (area = 0.53)
Precision-recall curve of class 13 (area = 0.55)
Precision-recall curve of class 14 (area = 0.16)
Precision-recall curve of class 15 (area = 0.06)
Precision-recall curve of class 16 (area = 0.03)
Precision-recall curve of class 17 (area = 0.27)

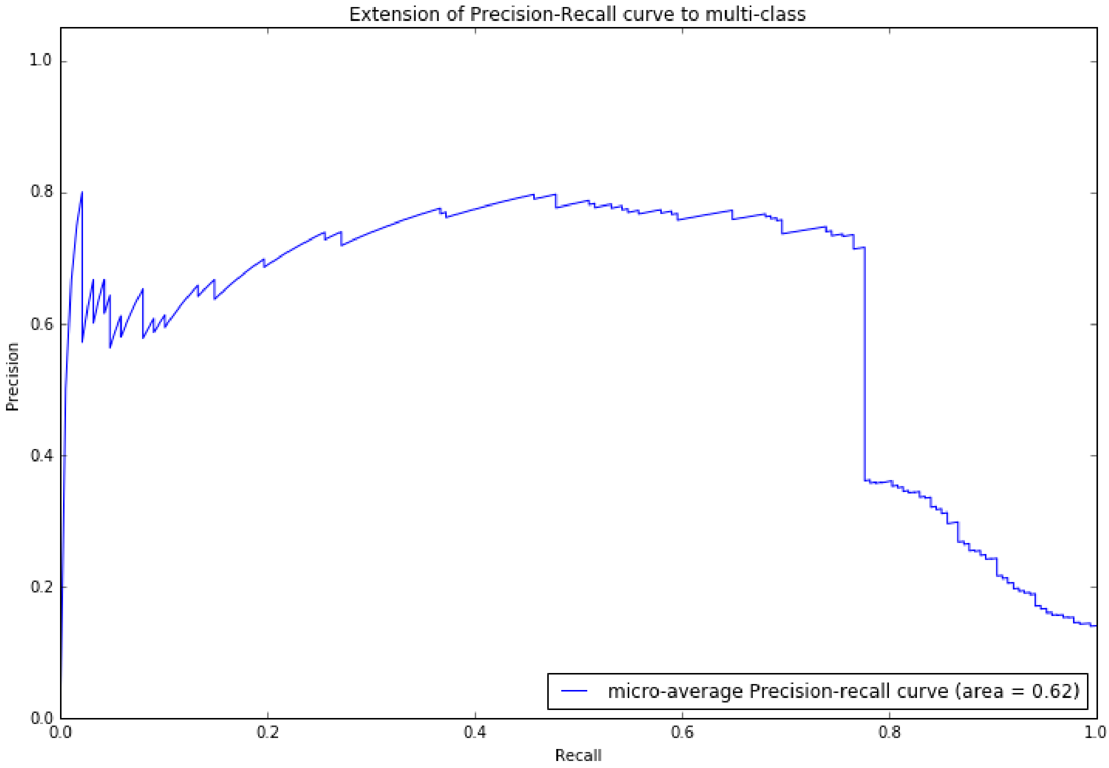
Environmental:

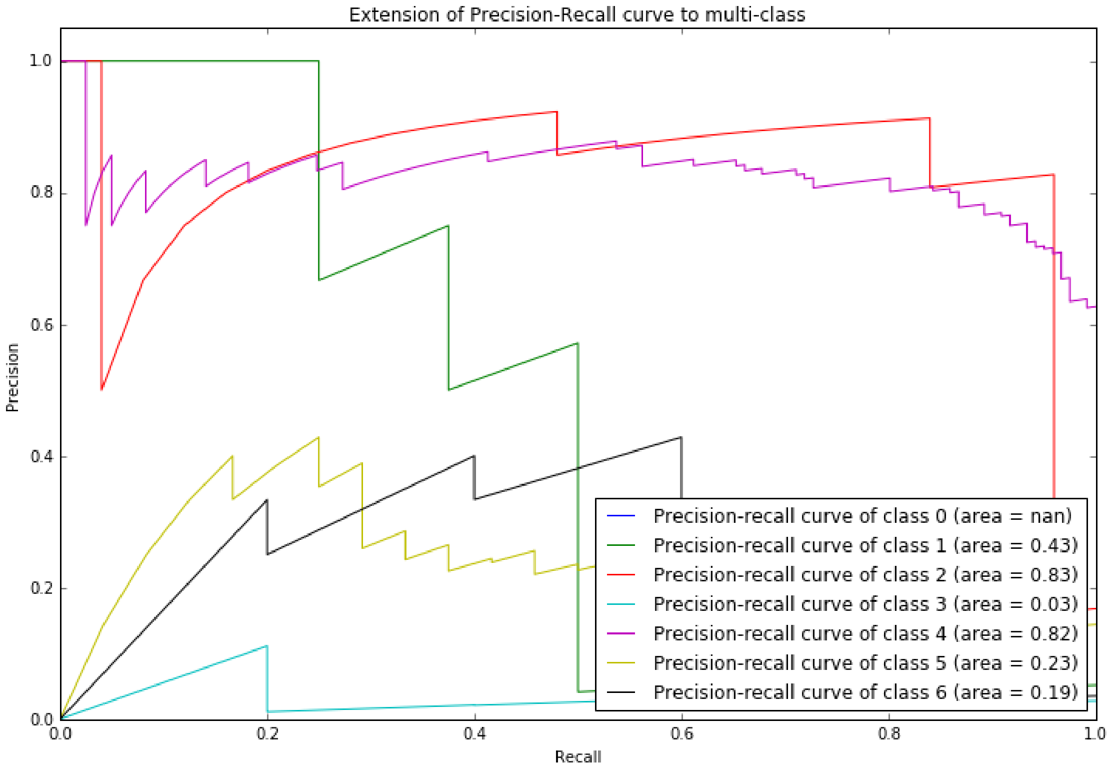
```
In [19]: svm_cv_roc(gene.data, gene.Stress)
```

```
The number of features selected for this variable is  25
```




The ROC curve for each class
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = nan)
ROC curve of class 1 (area = 0.66)
ROC curve of class 2 (area = 0.96)
ROC curve of class 3 (area = 0.45)
ROC curve of class 4 (area = 0.80)
ROC curve of class 5 (area = 0.70)
ROC curve of class 6 (area = 0.78)

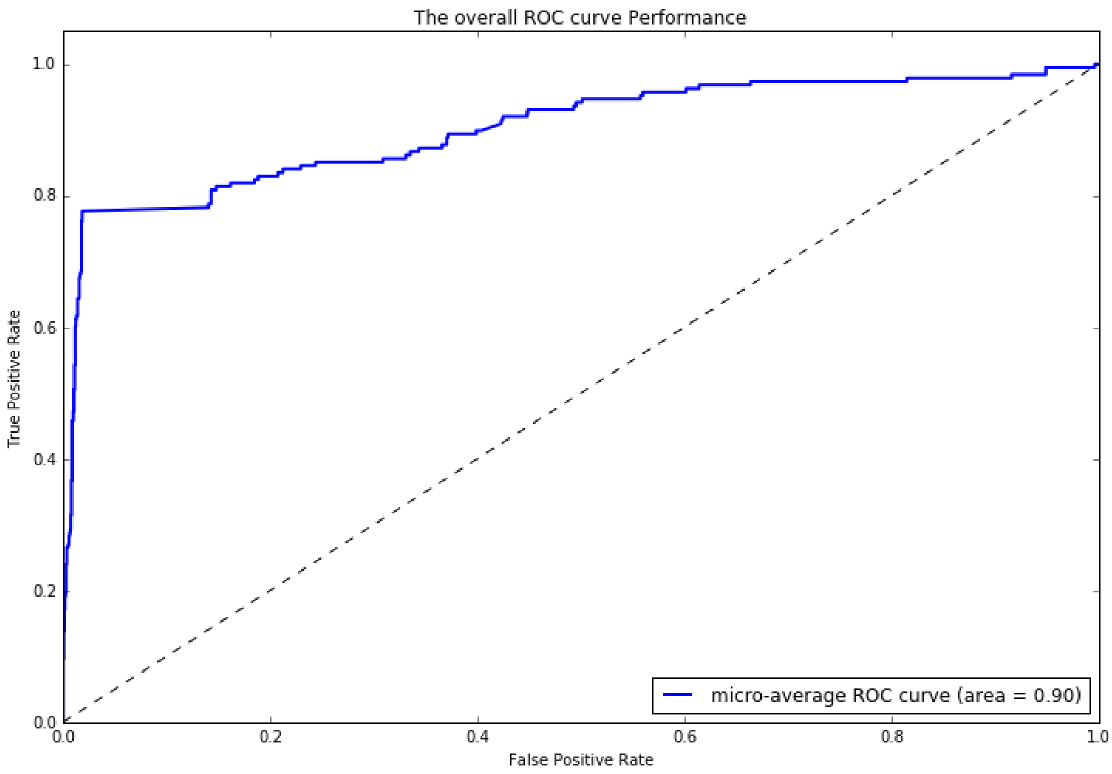

Extension of Precision-Recall curve to multi-class
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
micro-average Precision-recall curve (area = 0.62)

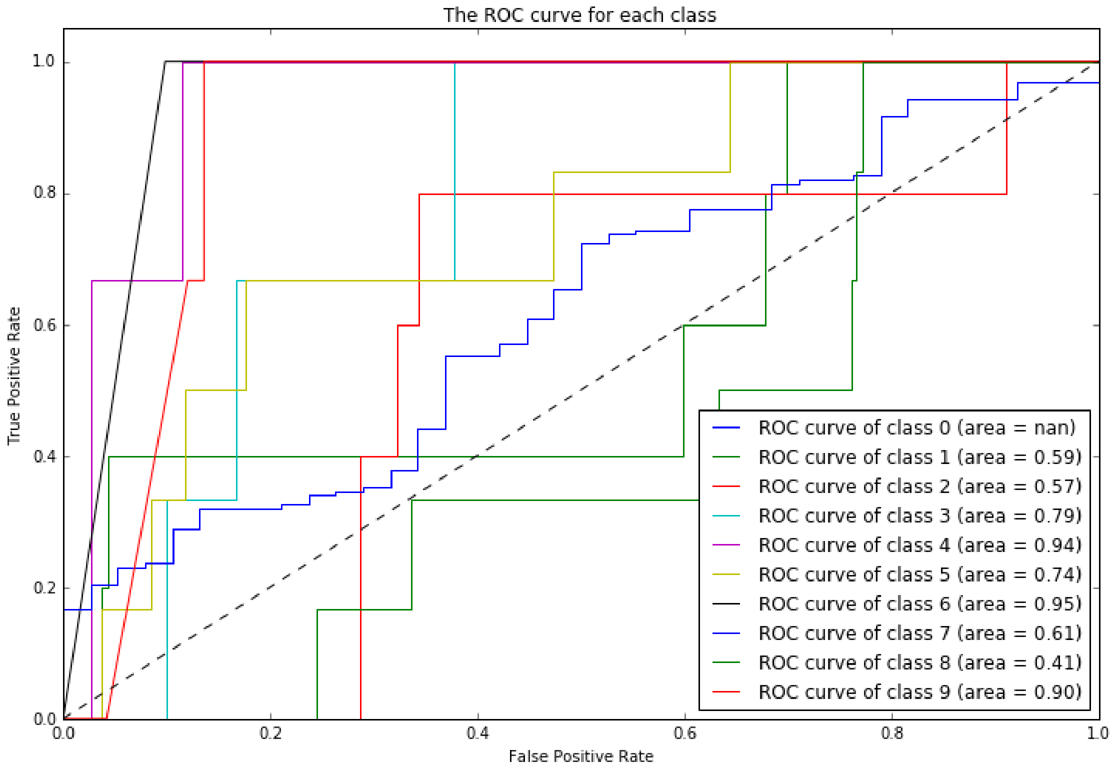


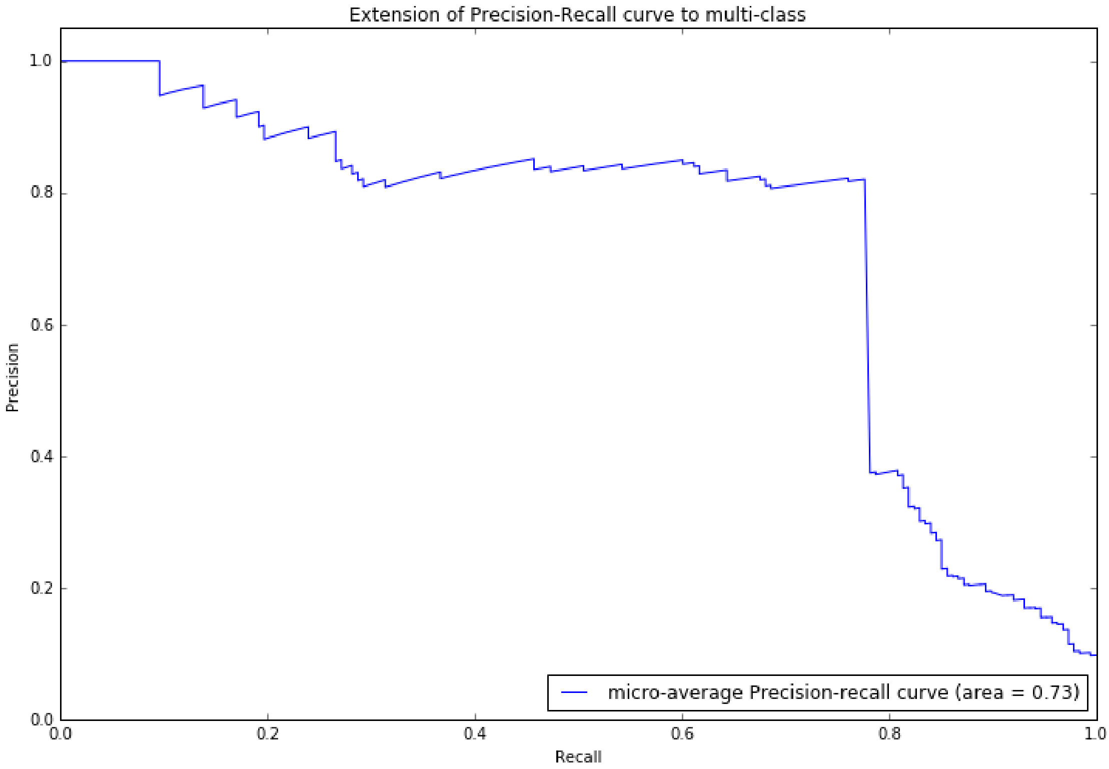
Gene Perturbation:

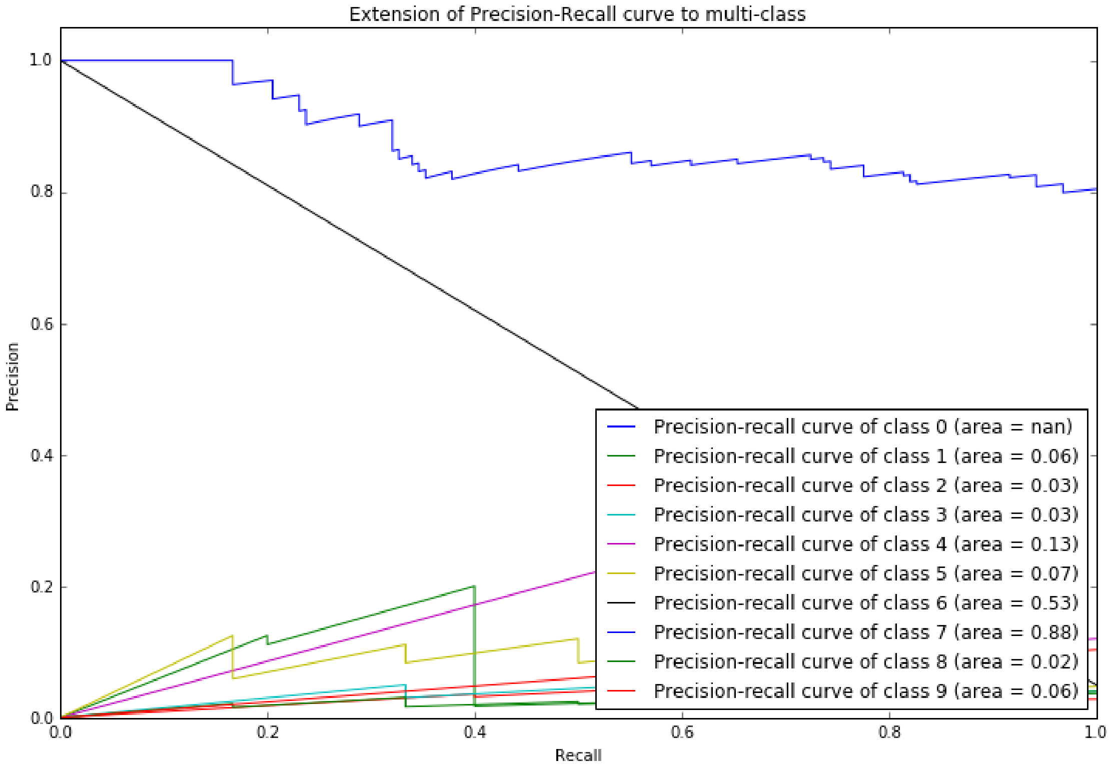
```
In [20]: svm_cv_roc(gene.data, gene.GenePerturbed)
```

```
The number of features selected for this variable is  10
```


The overall ROC curve Performance
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
micro-average ROC curve (area = 0.90)


The ROC curve for each class
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = nan)
ROC curve of class 1 (area = 0.59)
ROC curve of class 2 (area = 0.57)
ROC curve of class 3 (area = 0.79)
ROC curve of class 4 (area = 0.94)
ROC curve of class 5 (area = 0.74)
ROC curve of class 6 (area = 0.95)
ROC curve of class 7 (area = 0.61)
ROC curve of class 8 (area = 0.41)
ROC curve of class 9 (area = 0.90)


Extension of Precision-Recall curve to multi-class
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
micro-average Precision-recall curve (area = 0.73)

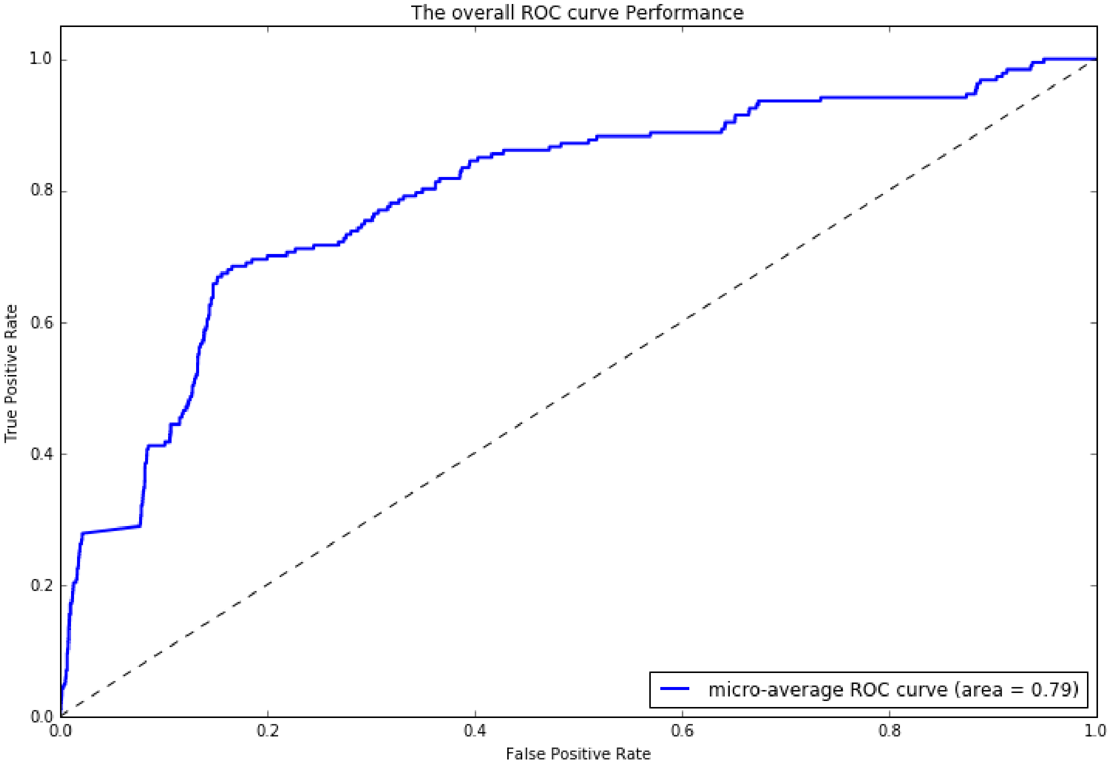

Extension of Precision-Recall curve to multi-class
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
Precision-recall curve of class 0 (area = nan)
Precision-recall curve of class 1 (area = 0.06)
Precision-recall curve of class 2 (area = 0.03)
Precision-recall curve of class 3 (area = 0.03)
Precision-recall curve of class 4 (area = 0.13)
Precision-recall curve of class 5 (area = 0.07)
Precision-recall curve of class 6 (area = 0.53)
Precision-recall curve of class 7 (area = 0.88)
Precision-recall curve of class 8 (area = 0.02)
Precision-recall curve of class 9 (area = 0.06)

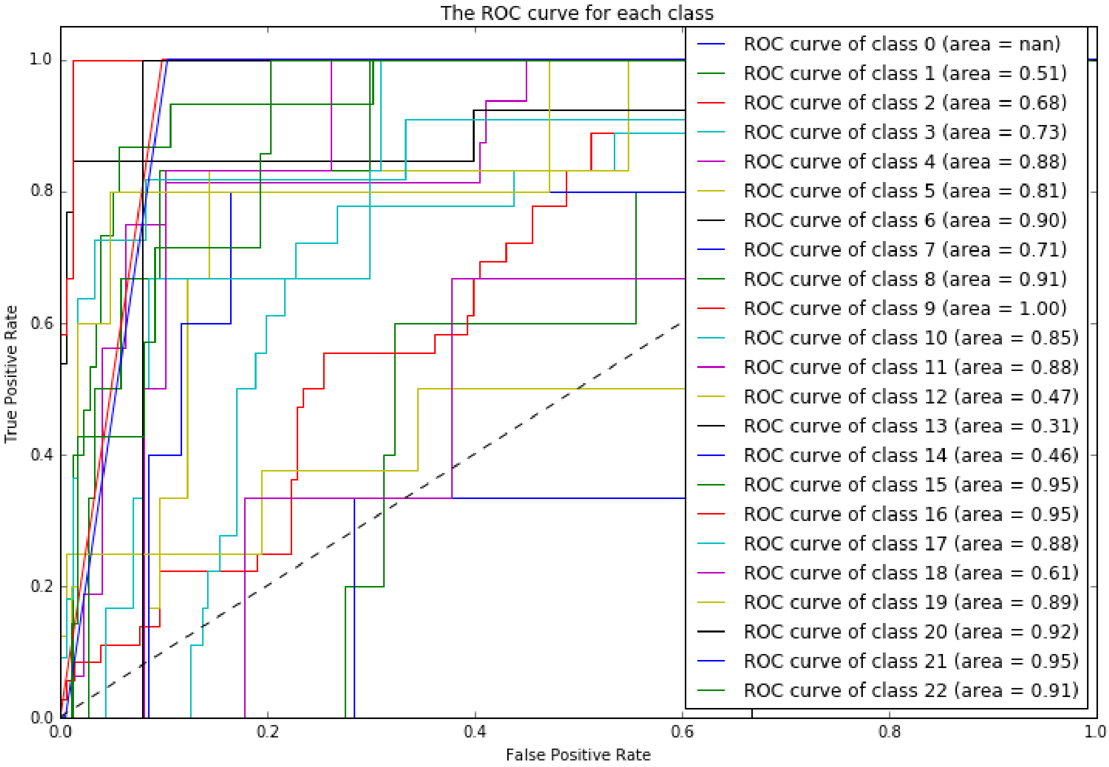
1.7 Question 5

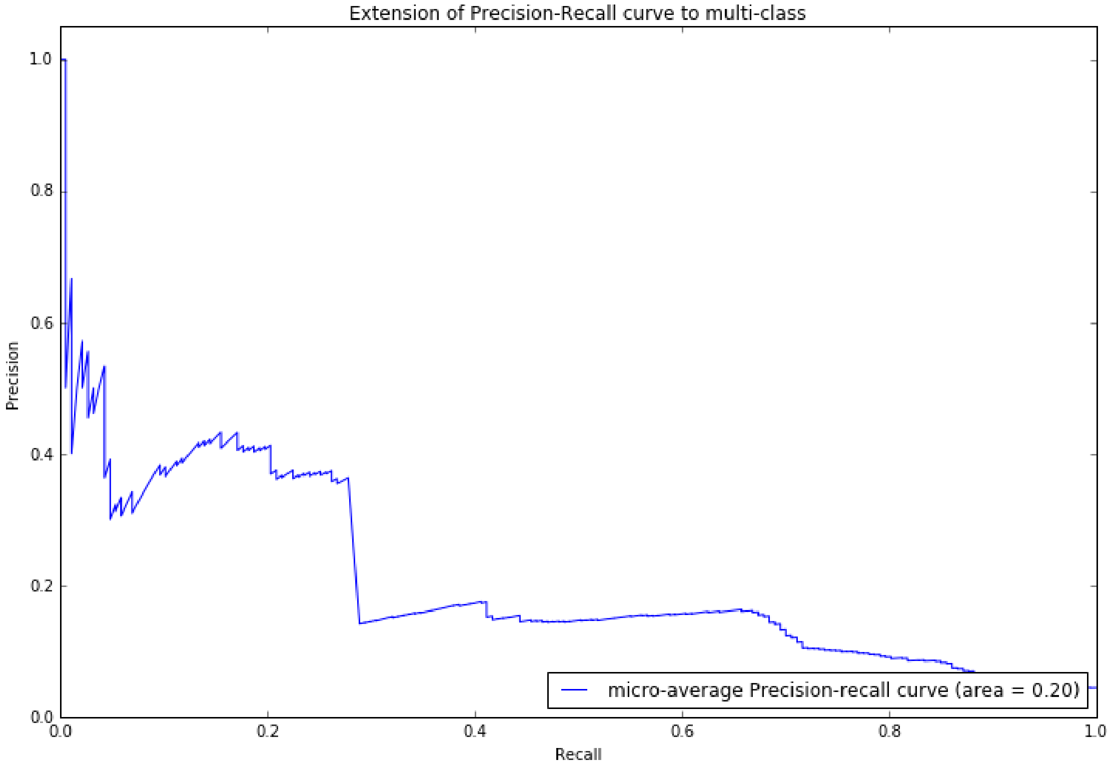
Since I have already done the combination of two variables of Medium and Stress in my R script, Now I can do the svm classification:

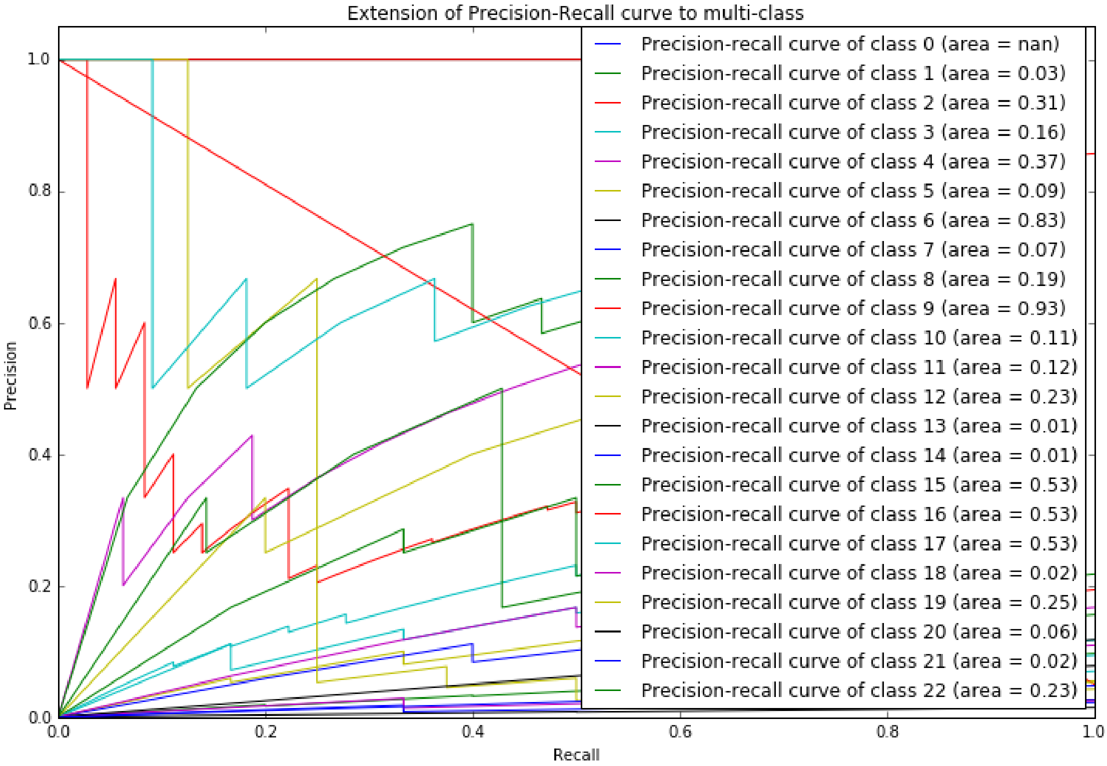
```
In [21]: svm_cv_roc(gene.data, gene.Medium_Stress)
```

```
The number of features selected for this variable is  25
```




The ROC curve for each class
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = nan)
ROC curve of class 1 (area = 0.51)
ROC curve of class 2 (area = 0.68)
ROC curve of class 3 (area = 0.73)
ROC curve of class 4 (area = 0.88)
ROC curve of class 5 (area = 0.81)
ROC curve of class 6 (area = 0.90)
ROC curve of class 7 (area = 0.71)
ROC curve of class 8 (area = 0.91)
ROC curve of class 9 (area = 1.00)
ROC curve of class 10 (area = 0.85)
ROC curve of class 11 (area = 0.88)
ROC curve of class 12 (area = 0.47)
ROC curve of class 13 (area = 0.31)
ROC curve of class 14 (area = 0.46)
ROC curve of class 15 (area = 0.95)
ROC curve of class 16 (area = 0.95)
ROC curve of class 17 (area = 0.88)
ROC curve of class 18 (area = 0.61)
ROC curve of class 19 (area = 0.89)
ROC curve of class 20 (area = 0.92)
ROC curve of class 21 (area = 0.95)
ROC curve of class 22 (area = 0.91)


Extension of Precision-Recall curve to multi-class
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
micro-average Precision-recall curve (area = 0.20)



The AUC are 0.84 and 0.87 individually, while the combination is 0.79. The AUPRC are 0.32 and 0.62 individually, while the combination is 0.20. These results indicate that this combination prediction is not as good as two individual predictors.

The baseline prediction performance is 0.84 for AUC, and 0.32 for AUPRC.

1.8 Question 6

I will first do PCA

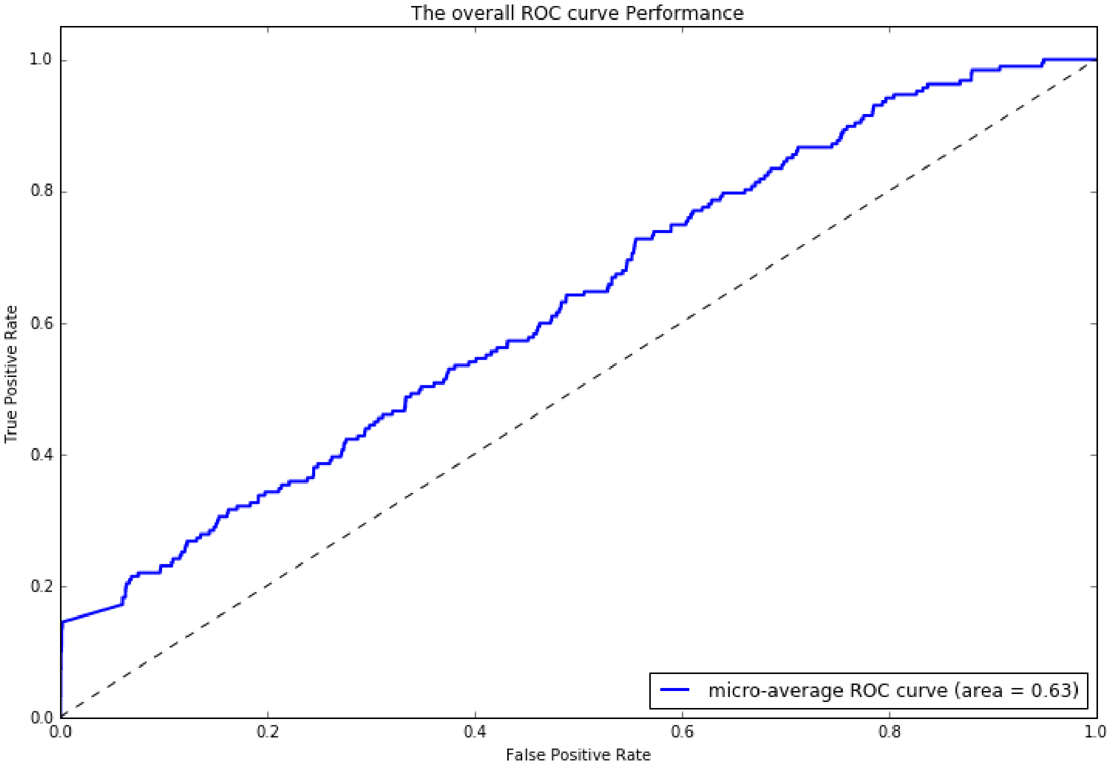
```
In [22]: from sklearn.decomposition import PCA
```

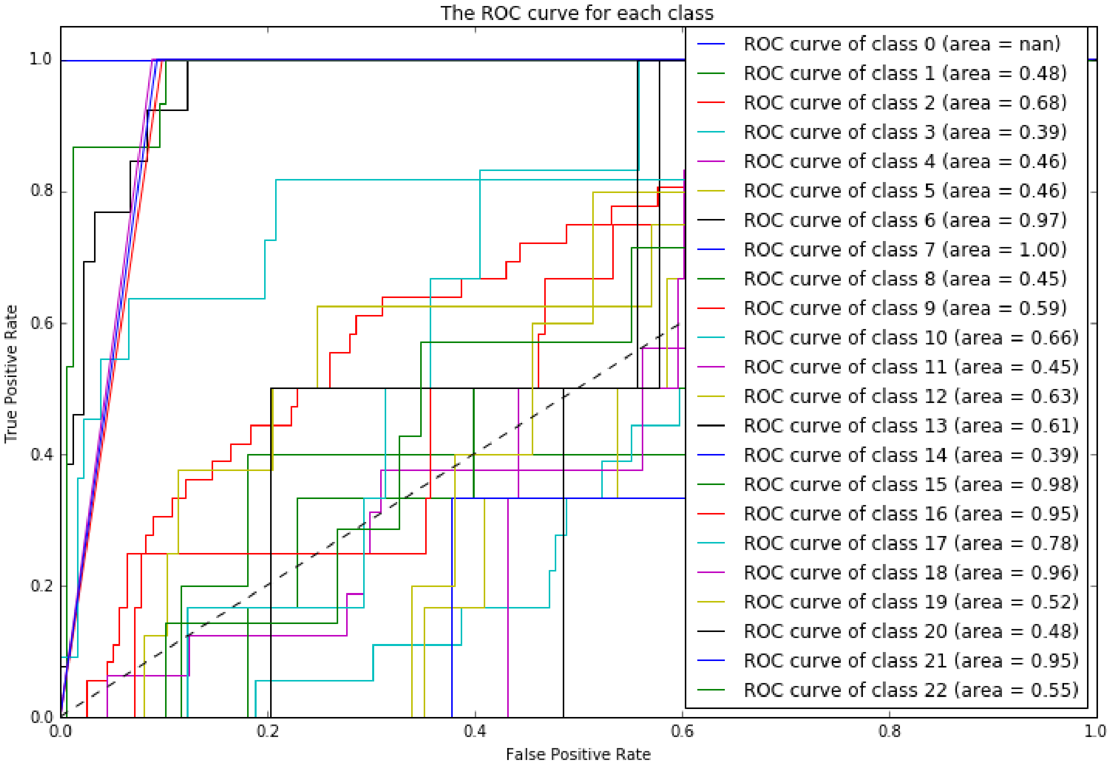
```
In [23]: # Normalize data first
         gene.data_pca = np.zeros_like(gene.data)
         for i in range(gene.data_pca.shape[1]):
             feature = gene.data.iloc[:, i]
             if np.std(feature) == 0:
                 gene.data_pca[:, i] = feature
             else:
                 gene.data_pca[:, i] = (feature - np.mean(feature)) / np.std(feature)
```

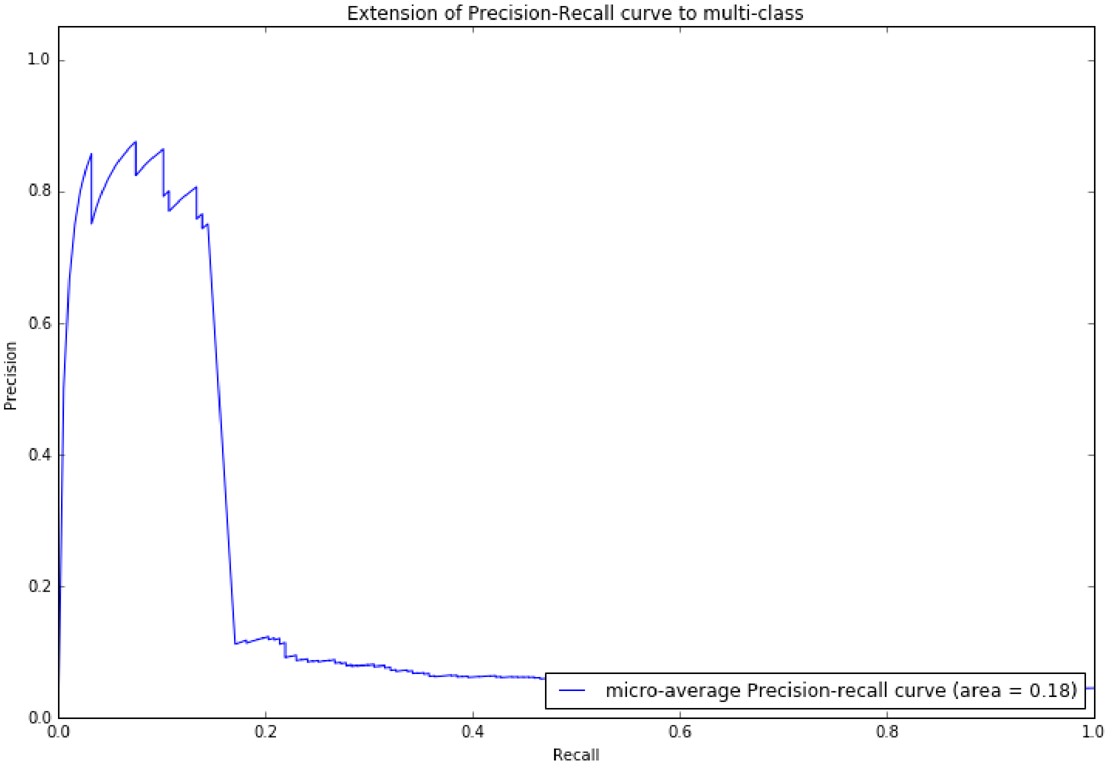
```
In [24]: # Set PCA parameter, the number is 3
         gene_pca = PCA(n_components = 3)
         gene_pca_fit = gene_pca.fit_transform(gene.data_pca)
```

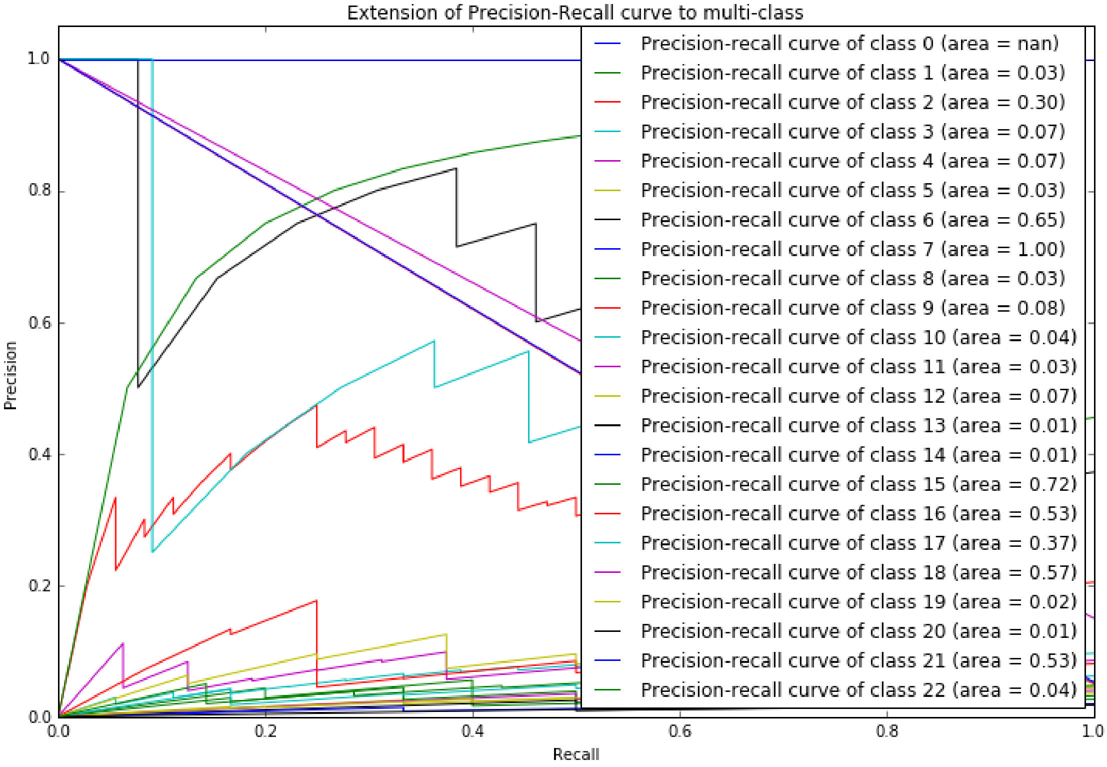
```
In [25]: svm_cv_roc(gene_pca_fit, gene.Medium_Stress)
```

```
The number of features selected for this variable is  3
```


The overall ROC curve Performance
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
micro-average ROC curve (area = 0.63)


The ROC curve for each class
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = nan)
ROC curve of class 1 (area = 0.48)
ROC curve of class 2 (area = 0.68)
ROC curve of class 3 (area = 0.39)
ROC curve of class 4 (area = 0.46)
ROC curve of class 5 (area = 0.46)
ROC curve of class 6 (area = 0.97)
ROC curve of class 7 (area = 1.00)
ROC curve of class 8 (area = 0.45)
ROC curve of class 9 (area = 0.59)
ROC curve of class 10 (area = 0.66)
ROC curve of class 11 (area = 0.45)
ROC curve of class 12 (area = 0.63)
ROC curve of class 13 (area = 0.61)
ROC curve of class 14 (area = 0.39)
ROC curve of class 15 (area = 0.98)
ROC curve of class 16 (area = 0.95)
ROC curve of class 17 (area = 0.78)
ROC curve of class 18 (area = 0.96)
ROC curve of class 19 (area = 0.52)
ROC curve of class 20 (area = 0.48)
ROC curve of class 21 (area = 0.95)
ROC curve of class 22 (area = 0.55)


Extension of Precision-Recall curve to multi-class
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
micro-average Precision-recall curve (area = 0.18)


Extension of Precision-Recall curve to multi-class
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
Precision-recall curve of class 0 (area = nan)
Precision-recall curve of class 1 (area = 0.03)
Precision-recall curve of class 2 (area = 0.30)
Precision-recall curve of class 3 (area = 0.07)
Precision-recall curve of class 4 (area = 0.07)
Precision-recall curve of class 5 (area = 0.03)
Precision-recall curve of class 6 (area = 0.65)
Precision-recall curve of class 7 (area = 1.00)
Precision-recall curve of class 8 (area = 0.03)
Precision-recall curve of class 9 (area = 0.08)
Precision-recall curve of class 10 (area = 0.04)
Precision-recall curve of class 11 (area = 0.03)
Precision-recall curve of class 12 (area = 0.07)
Precision-recall curve of class 13 (area = 0.01)
Precision-recall curve of class 14 (area = 0.01)
Precision-recall curve of class 15 (area = 0.72)
Precision-recall curve of class 16 (area = 0.53)
Precision-recall curve of class 17 (area = 0.37)
Precision-recall curve of class 18 (area = 0.57)
Precision-recall curve of class 19 (area = 0.02)
Precision-recall curve of class 20 (area = 0.01)
Precision-recall curve of class 21 (area = 0.53)
Precision-recall curve of class 22 (area = 0.04)

Compared with the results in Question 5, the AUC has been reduced from 0.79 to 0.63, and AUPRC decreases from 0.20 to 0.15, which means PCA preserves about 75% to 80% of the original information.