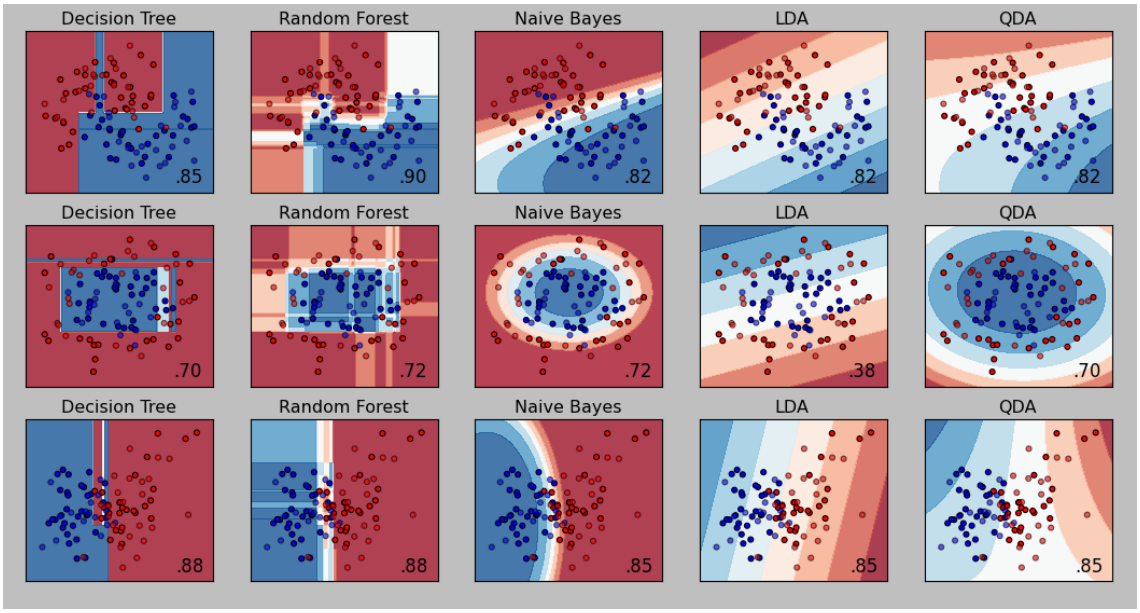
MODEL SELECTION PERFORMANCE ESTIMATION

Kyriakis Dimitris

June 2017

Methods in Bioinformatics



Contents

1 Introduction

In machine learning, Classification is a subcategory of supervised learning. Classification is the problem of identifying to which of a set of categories, a new observation belongs, on the basis of a training set of data containing observations whose category membership is known. Classification is one of the most widely used techniques in machine learning, for medical diagnosis and image classification etc. Linear classifiers are among the most practical classification methods. In this project, we have provided with 4 different Train datasets (Psoriasis, Lupus, Autism, Psoriasis RNAseq) and one dataset for each case in order to predict the labels of their samples. All data downloaded from https://inclass.kaggle.com/.

```
    Choose_Data_Set = input("""Choose_Data_Set:\n\t
              1.Lupus\n\t
              2.Psoriasis\n\t
              3.Autism\n\t
              4.Psoriasis_RNAseq\n\t
              (1/2/3/4): """)

    if Choose_Data_Set =="1":
        Train = Working_Dir + "Lupus/SRP062966Give.csv"
        Test = Working_Dir + "Lupus/SRP062966Validation.csv"
        Case= "Lupus"
        control = "control"

    elif Choose_Data_Set == "2":
        Train = "Psoriasis/GDS4602Give.csv"
        Test = "Psoriasis/GDS4602Validation.csv"
        Case= "Psoriasis"
        control = "healthy"

    elif Choose_Data_Set == "3":
        Train = Working_Dir + "Autism/GDS4431Give.csv"
        Test = Working_Dir + "Autism/GDS4431Validation.csv"
        Case= "Autism"
        control = "control"

    else:
        Train = Working_Dir + "Psoriasis_RNAseq/SRP035988Give.csv"
        Test = Working_Dir + "Psoriasis_RNAseq/SRP035988Validation.csv"
        Case= "Psoriasis_RNAseq"
        control = "control"
    print("\n\n#################\n"+Case + " Choosed\n")

    Data = np.genfromtxt(Train, delimiter=',', dtype=str)
    Test_Data_raw = np.genfromtxt(Test, delimiter=',', dtype=str)
    Test_Data_val = Test_Data_raw[1:,1:].astype(np.float) # VALUATION
    Test_Data_samples = Test_Data_raw[1:,0]
    Samples_Name = Data[1:,0].reshape(Data.shape[0]-1,1)
    Features = Data[0,1:]

    Train_Data = (Data[1:,1:Data.shape[1]-1]).astype(np.float) # TRAIN

    Labels_raw = Data[1:,Data.shape[1]-1]  != control
    Labels = Labels_raw.astype(int) # LABELS
    count_control = sum(Labels == 0)
    Min_Folds = min(count_control, Labels.shape[0] - count_control)

    print("\n\n#################\n"+"Data Loaded\n")
```

1.1 Model Selection

1.1.1 Cross Validation

Cross-validation, is a model validation technique for assessing the results of your classifier. It is mainly used ,when we want to estimate how accurately a predictive model will perform in practice. In a prediction problem, where a dataset of known data is given, we split our data in folds and every time we take the n-1 folds as training dataset and the one fold that left, as test dataset ("unknown data"). The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem) etc. An additive method to cross validation is **holdout method**. In hold out method, there is a split step before cross validation. We split the training set to three. The two-thirds for training and the other for estimating performance. Hold out method gives more unbiased results.

Common types of cross-validation:

- **Leave-p-out cross-validation (LpO CV)**
  Leave-p-out cross-validation involves using p observations as the validation set and the remaining observations as the training set. This is repeated on all ways to cut the original sample on a validation set of p observations and a training set.
- **Leave-one-out cross-validation (LOOCV)**
  Leave-one-out cross-validation is a particular case of leave-p-out cross-validation with $p = 1$.

1.1.2 Stratification

A better approach over the standard k-fold cross-validation is stratified k-fold cross-validation, which can yield better bias and variance estimates, especially in cases of unequal class proportions[3]. One common issue in data mining is the size of the data set. It is often limited. When this is the case, the test of the model is an issue. Usually, 2/3 of the data are used for training and validation and 1/3 for final testing. By chance, the training or the test set may not be representative of the overall data set. Consider for example a data set of 200 samples and 10 classes. It is likely that one of these 10 classes is not represented in the validation or test set.
To avoid this problem, you should take care of the fact that each class should be correctly represented in both the training and testing sets. This process is called stratification. One way to avoid doing stratification, regarding the training phase is to use k-fold cross-validation. Instead of having only one given validation set with a given class distribution, k different validation sets are used. However, this process does not guarantee a correct class distribution among the training and validation sets. So we can select the data in every fold based on the probability of a class.

```
1       kfold = StratifiedKFold(y=Labels, n_folds= Min_Folds, shuffle=True)
```

1.1.3 Nested-Cross Validation

A better way of using the holdout method for model selection is to separate the data into three parts: a training set, a validation set, and a test set. The training set is used to fit the different models, and the performance on the validation set is then used for the model selection. The advantage of having a test set that the model hasn't seen before during the training and model selection steps is that we can obtain a less biased estimate of its ability to generalize to new data.

The following figure illustrates the concept of holdout cross-validation where we use a validation set to repeatedly evaluate the performance of the model after training using different parameter values. Once we are satisfied with the tuning of parameter values, we estimate the models' generalization error on the test dataset[3]. This method called **nested cross-validation**. In figure we have an outer k-fold cross-validation loop to split the data into training and test folds, and an inner loop is used to select the model using k-fold cross-validation on the training fold[3].

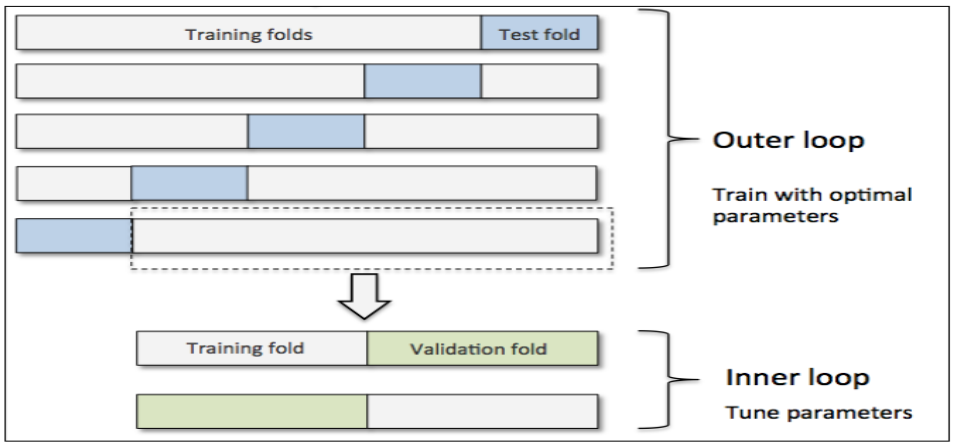


Figure 1: Nested cross-validation with five outer and two inner folds, which can be useful for large data sets where computational performance is important; this particular type of nested cross-validation is also known as 5x2 cross-validation [3].

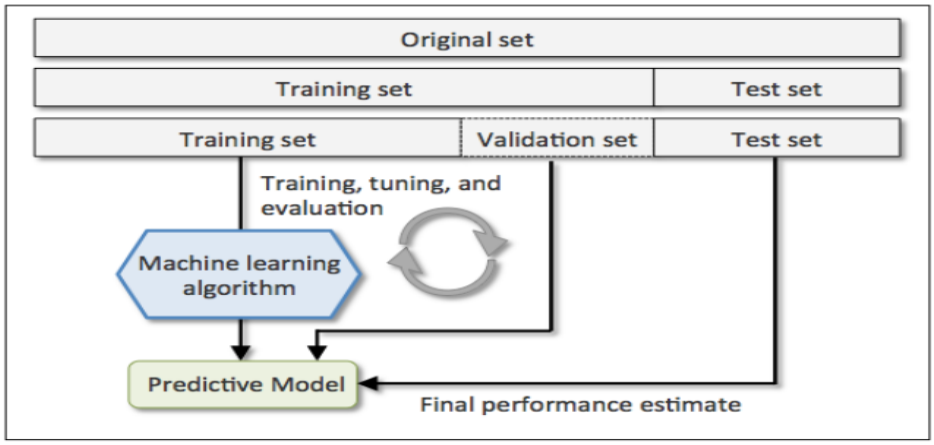


Figure 2: Nested cross validation method. First hide a test set. Then split the remain data to train and validation data in order to tune the hyperparameters and to choose the best model for our data[3].

According to some research, there are evidence in the machine learning, regarding whether N-fold cross-validation has better performance than LOOCV and vice-versa for binary classification[7]. Due to my pc performance couldn't run LOOCV on data.

Tuning algorithms: Model selection, must not violate "golden rule". In every loop we must be careful not to take information from data that we "do not know" yet. For that reason we constructed a pipeline to fit every time on training set Fig:2.

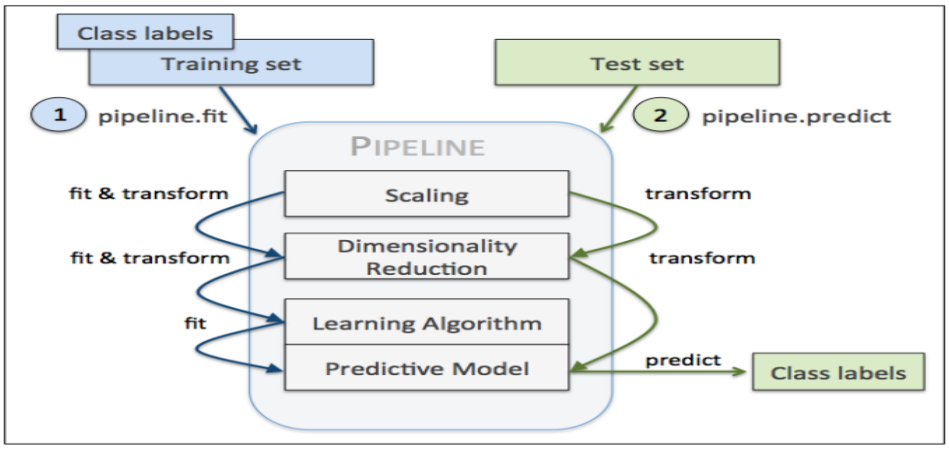


Figure 3: Pipeline created in order to fit it in every loop the exact steps[3].

A disadvantage of the holdout method is that the performance estimate is sensitive to how we partition the training set into the training and validation subsets. The estimate will vary for different samples of the data. A more robust technique for performance estimation, k-fold cross-validation, is to repeat the holdout method k times on k subsets of the training data[3]. My implementation of the above pipeline, is in methods section.

1.1.4 Ensembles of classifiers

Another method is combining classifiers in order to improve the performance of individual classifiers. These classifiers could be based on a variety of classification methodologies, and could achieve different rate of correctly classified individuals. The aim is to generate more certain, precise and accurate system results. This method can be used when the different classifiers give around the same accuracy. However ensembles, increasing storage requirements. The total storage depends on the size of each component classifier itself and the size of the ensemble (number of classifiers in the ensemble). The second weakness is increased computation: to classify an input query, all component classifiers (instead of a single classifier) must be processed, and thus it requires more execution time. The last weakness is decreased comprehensibility. Also, it is difficult to choose the classifiers that we must combine.

```
## Best hyperparameters ##
clf1 = LogisticRegression(penalty='l2',C= 0.0001)
clf2 = KNeighborsClassifier(metric='minkowski',n_neighbors=3)
clf3 = SVC(kernel = 'linear', C = 0.0001,probability=True)

# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),('clf1', clf1)])
pipe2 = Pipeline([('std', StandardScaler()),('clf2', clf2)])
pipe3 = Pipeline([('std', StandardScaler()),('clf3', clf3)])

## Ensemble ##
mv_clf =  VotingClassifier(estimators=[('clf1',pipe1),
                                       ('clf2', pipe2),
                                       ('clf3',pipe3)],
                                       voting='soft')
all_clf = [pipe1, pipe2, pipe3, mv_clf]
## Cross Validation ##
for clf, label in zip(all_clf, clf_labels):
    scores = cross_val_score(estimator=clf,X=inner_Train,y=inner_labels,cv=10,scoring='roc_auc
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"% (scores.mean(), scores.std(), label))

eAlg = VotingClassifier(estimators=[('clf1',pipe1),
                                    ('clf2', pipe2),
                                    ('clf3',pipe3)],
                                    voting='soft')
eAlg.fit(inner_Train,inner_labels)
y_test = Labels[test_idx]
pred_ens = eAlg.predict(inner_Test)
```

1.2 Limitations

Another major methodological concern is the problem of overfitting and underfitting. Overfitting is creating diagnostic models that may not generalize well to new data despite excellent performance on the training set. Since many algorithms are highly parametric and datasets consist of a relatively small number of high-dimensional samples, it is easy to overfit both the classifiers and the gene selection procedures especially when using intensive model search and powerful learners. As a result, the performance estimation is optimistic due to overfitting of the data. On the other hand, underfitting not learning characteristics that would generalize the model.

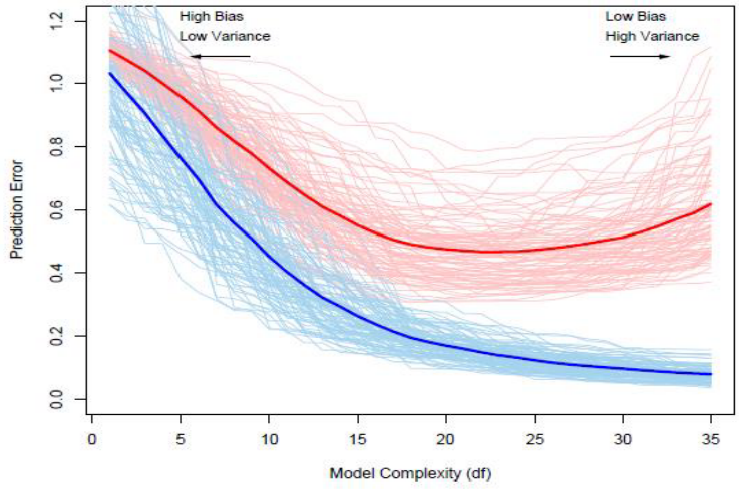


Figure 4: Behavior of test sample and training sample error as the model complexity is varied. the light blue curves show the training error, while the light red curves show the conditional test error for 100 training sets of size 50 each, as the model complexity is increased. The solid curve show the expected test error and the expected training error [**?**].

As the plot shows, the more the complexity increase the more accuracy we have until one threshold. In the plot above this threshold is around 20 number of complexity. After 20, the the CV gives us better results but our prediction in test data is falling.

1.3 Normalization-Scaling

Standardization (or Z-score normalization) is that the features will be rescaled so that they'll have the properties of a standard normal distribution with $\mu = 0$ and $\sigma = 1$ where μ is the mean (average) and σ is the standard deviation from the mean. Standard scores (also called z scores) of the samples are calculated as follows:

$$z = x - \mu\sigma$$

Standardizing the features so that they are centered around 0 with a standard deviation of 1 is not only important if we are comparing measurements that have different units, but it is also a general requirement for many machine learning algorithms.

An alternative approach to Z-score normalization (or standardization) is the so-called Min-Max scaling (often also simply called "normalization" - a common cause for ambiguities). In this approach, the data is scaled to a fixed range - usually 0 to 1. The cost of having this bounded range - in contrast to standardization - is that we will end up with smaller standard deviations, which can suppress the effect of outliers. A Min-Max scaling is typically done via the following equation:

$$X_{norm} = X - X_{min}/X_{max}X_{min}$$

In clustering analyses, standardization may be especially crucial in order to compare similarities between features based on certain distance measures[3].

1.4 Feature selection

Dimensionality reduction can improve classification performance[4].Feature selection is different from dimensionality reduction. Both methods seek to reduce the number of attributes in the dataset, but a dimensionality reduction method do so by creating new combinations of attributes, where as feature selection methods include and exclude attributes present in the data without changing them[2]. Feature selection methods aid you in your mission to create an accurate predictive model. They help you by choosing features that will give you as good or better accuracy whilst requiring less data.

Feature selection methods can be used to identify and remove unneeded and redundant features from data that do not contribute to the accuracy of a predictive model or may in fact decrease the accuracy of the model. Fewer attributes is desirable because it reduces the complexity of the model, avoiding over-fitting.

The objective of variable selection is three-fold: **improving the prediction performance** of the predictors, providing faster and **more cost-effective predictors**, and providing a better understanding of the underlying process that generated the data.

I used feature selection based on the variance of every feature. Tested for different thresholds 0.00001-0.01 but the results was the same for every classifier that tested. Because of the computational cost I could not perform it for all datasets.

1.5 Estimation

The aim of the nested cross validation, is to estimate our prediction in a new dataset. There are different statistics in order to evaluate the accuracy of our classifier predictions. Accuracy is a statistic but not a reliable metric for the real performance of a classifier, because it will yield misleading results if the data set is unbalanced (that is, when the number of samples in different classes vary greatly). For example, if there were 95 patients and only 5 healthy in the data set, the classifier could easily be biased into classifying all the samples as patients. The overall accuracy would be 95%, but in practice the classifier would have a 100% recognition rate for the class with the patients but a 0% recognition rate for the control class.

For that reason, there was need in machine learning for more robust estimations. A confusion matrix (error matrix), is a visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class (or vice versa)[9].

It is a special kind of contingency table, with two dimensions ("actual" and "predicted"), and identical sets of "classes" in both dimensions (each combination of dimension and class is a variable in the contingency table).

With Confusion matrix in our hand, we can compute different statistics about our classification Fig[5].

		predicted condition			
	total population	prediction positive	prediction negative	Prevalence = $\frac{\Sigma \text{ condition positive}}{\Sigma \text{ total population}}$	
true condition	condition positive	**True Positive (TP)**	**False Negative (FN)** (type II error)	True Positive Rate (TPR), Sensitivity, Recall, Probability of Detection = $\frac{\Sigma \text{ TP}}{\Sigma \text{ condition positive}}$	False Negative Rate (FNR), Miss Rate = $\frac{\Sigma \text{ FN}}{\Sigma \text{ condition positive}}$
	condition negative	**False Positive (FP)** (Type I error)	**True Negative (TN)**	False Positive Rate (FPR), Fall-out, Probability of False Alarm = $\frac{\Sigma \text{ FP}}{\Sigma \text{ condition negative}}$	True Negative Rate (TNR), Specificity (SPC) = $\frac{\Sigma \text{ TN}}{\Sigma \text{ condition negative}}$
	Accuracy = $\frac{\Sigma \text{ TP} + \Sigma \text{ TN}}{\Sigma \text{ total population}}$	Positive Predictive Value (PPV), Precision = $\frac{\Sigma \text{ TP}}{\Sigma \text{ prediction positive}}$	False Omission Rate (FOR) = $\frac{\Sigma \text{ FN}}{\Sigma \text{ prediction negative}}$	Positive Likelihood Ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic Odds Ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
		False Discovery Rate (FDR) = $\frac{\Sigma \text{ FP}}{\Sigma \text{ prediction positive}}$	Negative Predictive Value (NPV) = $\frac{\Sigma \text{ TN}}{\Sigma \text{ prediction negative}}$	Negative Likelihood Ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$	

Figure 5: **Confusion Matrix:** Table with two rows and two columns that reports the number of false positives, false negatives, true positives, and true negatives.

In the results only the three below estimation will be referred.

Accuracy (ACC): is the proportion of true results (both true positives and true negatives) among the total number of cases examined.

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{P + N} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

False discovery rate (FDR):the rate of type I errors in null hypothesis testing when conducting multiple comparisons.

$$\text{FDR} = \frac{\text{FP}}{\text{FP} + \text{TP}} = 1 - \text{PPV}$$

F1 score: Is a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0. is the harmonic mean of precision and sensitivity

$$F_1 = 2 \cdot \frac{\text{PPV} \cdot \text{TPR}}{\text{PPV} + \text{TPR}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$

1.6 Supervised Classifiers

1.6.1 K-Nearest Neighbours (KNN)

In KNN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. Finally, kNN is powerful because it does not assume anything about the data, other than a distance measure can be calculated consistently between any two instances. The distance metric that we will in Knn depends on our data structure.
If classes are less than 10 and data are not sparse we can use :

- Euclidean
- Manhattan
- Minkowski
- Chebychev

On the other hand if we have more than 10 classes and our data is sparse:

- Mahalanobis
- Cosine
- Correlation

For this project, the hyperparameters of KNN that used was Euclidean, Minkowski and Manhattan distance metrics. Every distance was checked for $k = 1 : 20$ neighbours and also for non-weighted and weighted (according to the distance) vote.

1.6.2 Logistic Regression

Logistic regression is one of the most popular machine learning algorithms for binary classification. This is because it is a simple algorithm that performs very well on a wide range of problems. The logistic function is the core of the logistic regression technique. The logistic function is:

$$transformed = 1/(1 + e^{-}x)$$

e: numerical constant Euler's number
x: is a input
The logistic function will transform all the inputs into the range [0, 1]. Also, this function, gives us the ability, as long as our mean value is zero, to give as input positive and negative values and always get out a consistent transform into the new range. The logistic regression model takes real-valued inputs and makes a prediction as to the probability of the input belonging to the default class (class 0). If the probability is ¿ 0.5 we can take the output as a prediction for the default class (class 0), otherwise the prediction is for the other class (class 1). For this dataset, the logistic regression has three coefficients just like linear regression, for example:

$$output = b0 + b1 * x1 + b2 * x2$$

The aim of the logistic regression learning algorithm will be to find the best values for the coefficients (b0, b1 and b2) based on the training data. Unlike linear regression, the output is transformed into a probability using the logistic function:

$$p(class = 0) = 1/(1 + e^{(} - output))$$

The hyperparameters that tested for this classifier are the penalty and the cost. Ordinary linear regression (L^2) is used to minimize total squared error, while "robust" (L^1) regression minimizes the total absolute value of error.

1.6.3 Support Vector Machine (SVM)

Another powerful and widely used learning algorithm, in recent years, is the support vector machine (SVM), which can be considered as an extension of the perceptron. Using the perceptron algorithm, we minimized misclassification errors. However, in SVMs, our optimization objective is to maximize the margin. The margin is defined as the distance between the separating hyperplane (decision boundary) and the training samples that are closest to this hyperplane, which are the so-called support vectors [2]. More specific, the hyperplane is based on a set of boundary training instances, called support vectors. Support Vector Machines (SVMs) map the data to a higher dimensional space via a kernel function and then identify the maximum-margin hyperplane in order to separate training instances[8]. New samples are classified based on the side of the hyperplane they fall into. The optimization problem is most often formulated in a way that allows for non-separable data by penalizing misclassifications[8].

SVMs according to some researches, have seen that achieve better classification performance than other learning algorithms. Also, SVMs seems to be insensitive in dimensionality and handle very large-scale classification in both sample and variables [4].

SVMs in the beginning could only be applied to binary classification problems. Multicategory classification is significantly harder than binary, so in the last few years, were created SVMs that allow alll types, binary or multicategory [6].In our case, we have to classify dataset with 2 classes. So, we will use Binary SVMs.

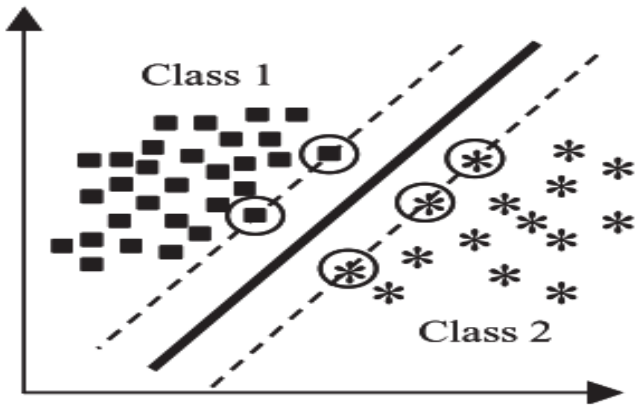


Figure 6: A binary SVM selects a hyperplane (bold line) that maximizes the width of the 'gap' (margin) between the two classes. The hyperplane is spe- cified by 'boundary' training instances, called support vectors shown with circles. New cases are classified according to the side of the hyperplane they fall into. [4].

2 Method

This project implemented in **python 3.6.0**. The main library that was used is**sklearn (0.18.1)**. We used the Standard scaler for scaling the data.

The variance of each feature was choose as a parameter for feature selection. The threshold for variances tested in range 0.000001-0.1.

```
#================================================================================#
#============================== Choose Classifier ===============================#
#================================================================================#
clas = input("Choose Classifier\n1. KNN\n2.Logistic Regression\n3. SVM\n((1/2/3) = ")

def Classifier_Ch(choice_cl,Min_F,thres):
     '''
    **Description:**\n
    Create a pipeline with stdscale, variance feature selection and classifier (user input).\n
    **Input:**\n
    - Classifier: 1 = KNN, 2 = Logistic Regression, 3. SVM\n
    - Min_f: Number of Folds\n
    - Variance threshold\n
    **Output:**\n
    - Pipeline
    '''
    ## Assign Scaler  ##
    scaler = StandardScaler()

    ## Assign Feature selection ##
    # Tested :selector = VarianceThreshold(threshold = 0.00001)

    ## Assign Classifier ##
    if choice_cl == "1" :
        clf = KNeighborsClassifier()
        param_grid = ...
    elif choice_cl == "2" :
        clf = LogisticRegression()
        param_grid = ...
    else:
        clf = SVC()
        param_grid = ...

    ## Create Pipeline ##
    pipe = Pipeline([('std', scaler),('feat', selector),('clf', clf)])
    gcv = GridSearchCV(estimator=pipe,param_grid=param_grid,scoring='accuracy',cv=Min_F)
    return gcv
```

In this implementation, I used the KNN, Logistic Regression and SVM as classifiers.
For ever classifier, tested several combinations of their hyperparameters.

K-Nearest Neighbours (KNN):

```
if choice_cl == "1" :
    clf = KNeighborsClassifier()
    param_grid = [{'clf__n_neighbors': list(range(1, 10)),
                'clf__metric': ['minkowski','euclidean'] ,
               'clf__weights':['uniform','distance']}]
```

K-Nearest Neighbours (KNN)										
Number of Neighbours	1	2	3	4	5	6	7	8	9	10
Metric	Euclidean	Minkowski								
Weights	uniform	distance								

Logistic Regression:

```
elif choice_cl == "2" :
    clf = LogisticRegression()
    param_grid = [{'clf__penalty': ['l1','l2'],
                'clf__C': np.power(10., np.arange(-4, 4))}]
```

Logistic Regression							
Penalty	L1	L2					
Cost	0.0001	0.001	0.01	0.1	10	100	1000

Support Vectror Machine (SVM):

```
else:
    clf = SVC()
    param_grid = [{'clf__kernel': ['rbf'],
                        'clf__C': [0.0001, 0.001, 0.01, 0.1, 10, 100],
                    'clf__gamma': [0.001, 0.0001]},
                  {'clf__kernel': ['linear'],
                      'clf__C': [0.0001, 0.001, 0.01, 0.1, 10, 100]}]
```

Support Vectro Machine (SVM						
Kernel	Linear	RBF				
Cost	0.0001	0.001	0.01	0.1	10	100
gamma	0.0001	0.001				

Repeated Nested-cross Validation used to estimate the accuracy of our classification pipeline.

```
## Repeated ##
for i in [1,2,3,4,5]:
    ## Nested-Cross Val ##
    kfold = StratifiedKFold(y=Labels, n_folds= Min_Folds, shuffle=True, random_state =1)
    mean_acc = 0
    Best_Param = {}

    for train_idx, test_idx in kfold:
        ## Assign ##
        inner_Train = Train_Data[train_idx]
        S = inner_Train.shape[0]
        D = inner_Train.shape[1]
        inner_Test = Train_Data[test_idx]
        inner_labels = Labels[train_idx]
        count_control_inner = sum(inner_labels == 0)
        Min_Folds_inner = min(count_control_inner, inner_labels.shape[0] - count_control_inner)
        if Min_Folds_inner >=10:
            Min_Folds_inner = 9
        ## Prepare ##
        gcv = Classifier_Ch(clas ,Min_Folds_inner,thres)
        gcv.fit(inner_Train,inner_labels)
        best_par = gcv.best_params_
        print(best_par)
        if str(best_par) in Best_Param.keys():
            Best_Param[str(best_par)] +=1
        else:
            Best_Param[str(best_par)] = 1
        ## Predict ##
        y_pred = gcv.predict(inner_Test)
        ## Accuracy ##
        acc = accuracy_score(y_true=Labels[test_idx], y_pred=y_pred)
        print(' | inner ACC %.2f%% | outer ACC %.2f%%' % (gcv.best_score_ * 100, acc * 100))
        mean_acc = mean_acc + acc


    print(mean_acc/Min_Folds)
```

Then, we examine the best methods in the outer loop in order to choose the most suitable parameters. Finally, we use these parameters to predict the labels from the validation Data-set.

3 Results

After the repeated nested cross validation, I used the best hyperparameters for each classifier and construct confusion matrices. Then using ensemble classification method, create a major vote confusion matrix.

3.1 Autism

The repeated nested cross validation, gave us that the best hyperparameters form Autism dataset are:

Autism: Hyperparameters For Classifiers			
SVM	Kernel = Linear	Cost = 0.0001	
KNN	Metric = Minkowski	N Neighbors = 7	Weights = Uniform
Logistic Regression	Penalty = l2	Cost = 0.001	

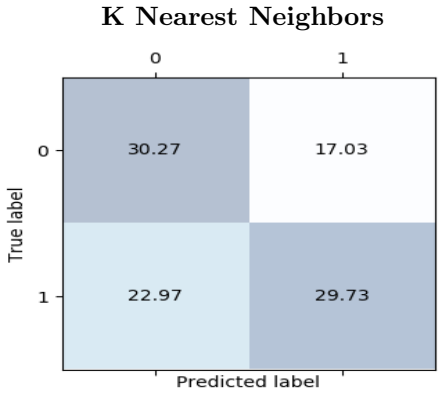


Figure 7: *Accuracy* = 60.00%, $FDR = 36.01\%$, $F1 = 60.21\%$.

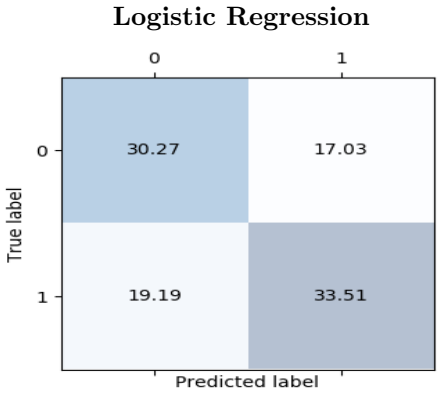


Figure 8: *Accuracy* = 63.78%, $FDR = 36.0\%$, $F1 = 62.57\%$.

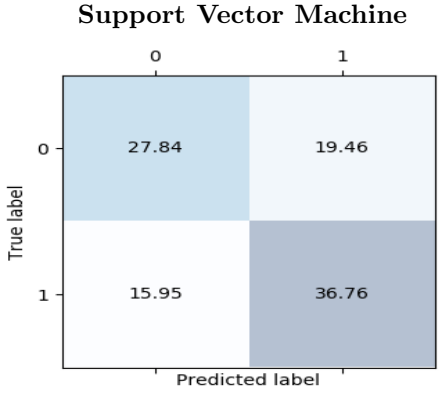


Figure 9: *Accuracy* = 64.6%, $FDR = 41.14\%$, $F1 = 61.13\%$.

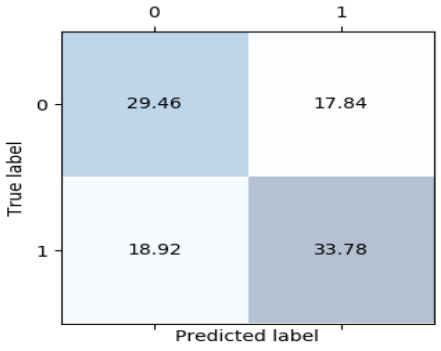


Figure 10: *Accuracy* = 63.24%, $FDR = 37.72\%$, $F1 = 61.58\%$.

As we can see, from the plots below, We can tell for certain which classifier is the best because of the large std. After the repeated nested we estimate that the accuracy of our predictions is around 63%. We can try ensemble majority vote, because classifiers have around the same accuracies and may increase our accuracy of predictions. We estimate that with Ensemble we have 63.2% +-11% accuracy so we expect around $(34.56 - 18.92$ errors. The results of Kaggle show that I made 27 wrong predictions. It is in the interval that I predict.

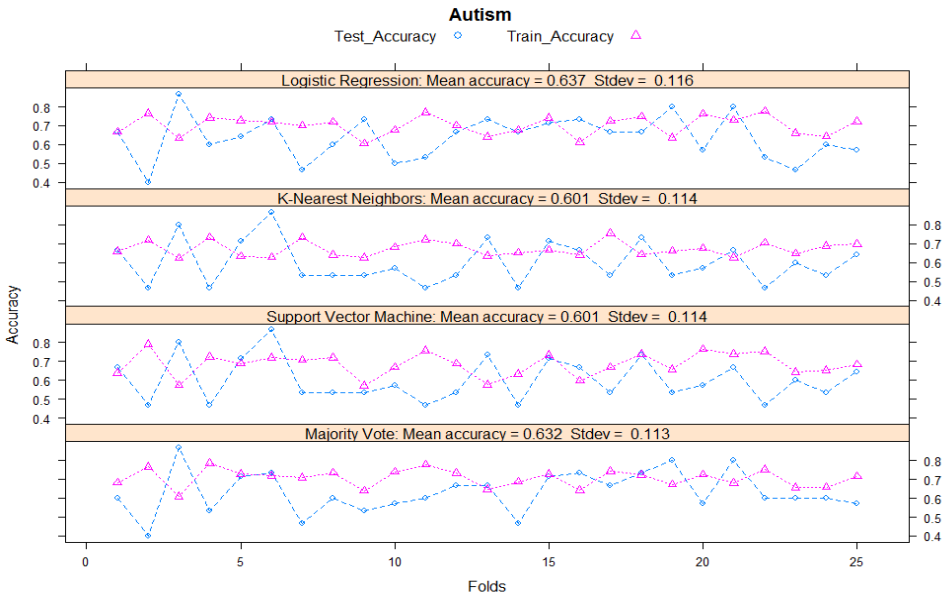


Figure 11: Autism.

3.2 Lupus

The repeated nested cross validation, gave us that the best hyperparameters for Lupus dataset:

Lupus: Hyperparameters For Classifiers			
SVM	Kernel = Linear	Cost = 0.0001	
KNN	Metric = Manhattan	N Neighbors = 5	Weights = Uniform
Logistic Regression	Penalty = l2	Cost = 10	

As we can see, from the plots below, SVM had the best accuracy. After the repeated nested we estimate that the accuracy of our predictions is around 97,63%. It has only 2,37 % error rate, and this is a conserved estimation. In this case, we must not trust ensemble majority vote, because classifiers have a large difference between their accuracies. Using SVM we expect around 2,3% errors. Lupus have 58 samples, so we expect that we will have 1,3 wrong predictions (conservative estimation). This is confirmed by Kaggle results, equal to zero.

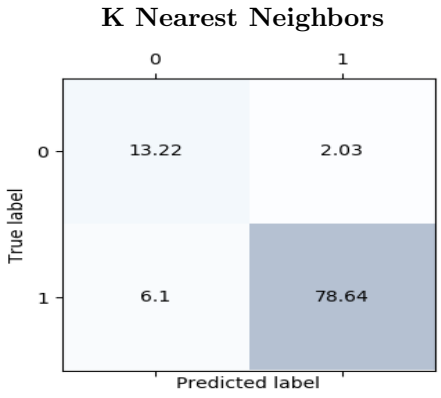


Figure 12: $Accuracy = 90.86\%$, $FDR = 13.31\%$, $F1 = 76.48\%$.

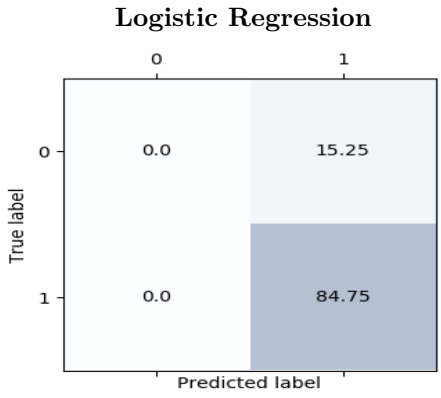


Figure 13: $Accuracy = 84.75\%$, $FDR = 100.0\%$, $F1 = 0\%$.

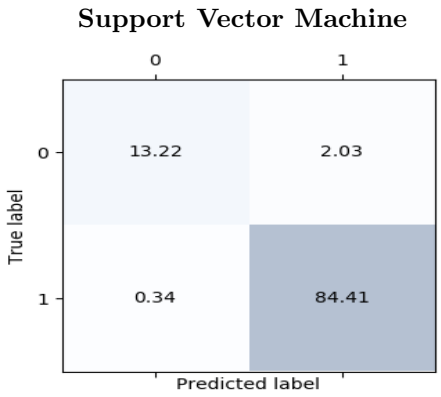


Figure 14: $Accuracy = 97.63\%$, $FDR = 13.31\%$, $F1 = 91.77\%$.

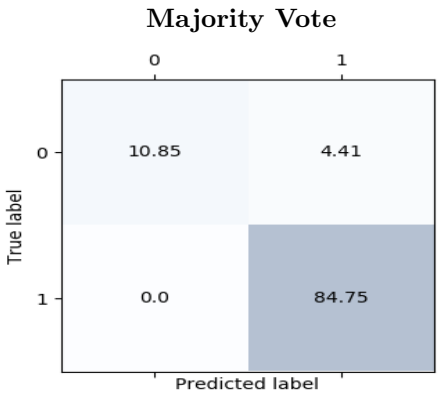


Figure 15: $Accuracy = 95.59\%$, $FDR = 28.9\%$, $F1 = 83.11\%$.

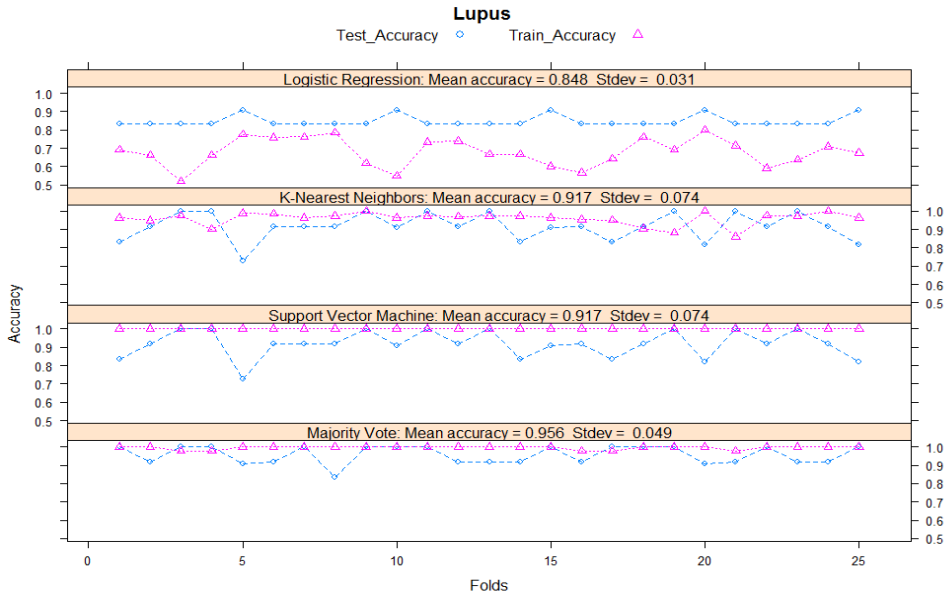


Figure 16: Lupus.

3.3 Psoriasis

The repeated nested cross validation, gave us that the best hyperparameters for Psoriasis dataset are:

Psoriasis: Hyperparameters For Classifiers			
SVM	Kernel = Linear	Cost = 0.0001	
KNN	Metric = Minkowski	N Neighbors = 1	Weights = Uniform
Logistic Regression	Penalty = l1	Cost = 100	

Knn seems the best classifier for this dataset, butthe diffeerence between the classifiers is to small. In this case may we could use Ensemble in order to minimize error II. We estimated that the we will have around $23.4 - 16.2$ miss classified samples. The result of my predictions was 17 errors something that we expected.

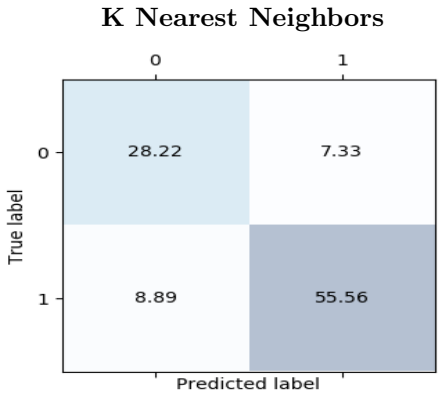


Figure 17: *Accuracy* = 83.78%, $FDR = 20.62\%$, $F1 = 77.68\%$.

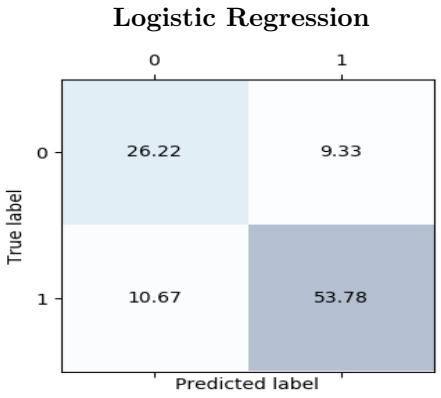


Figure 18: *Accuracy* = 80.0%, $FDR = 26.24\%$, $F1 = 72.39\%$.

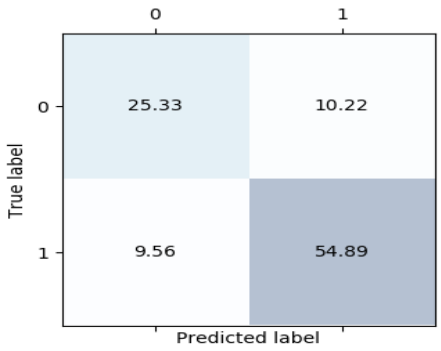


Figure 19: *Accuracy* = 80.22%, $FDR = 28.75\%$, $F1 = 71.92\%$.

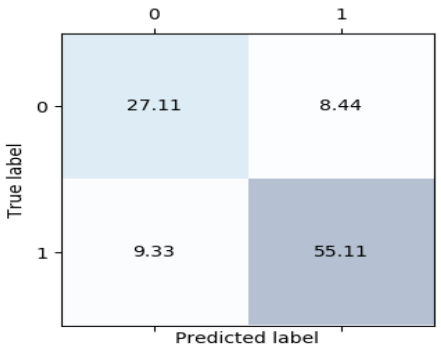


Figure 20: *Accuracy* = 82.22%, $FDR = 23.74\%$, $F1 = 75.32\%$.

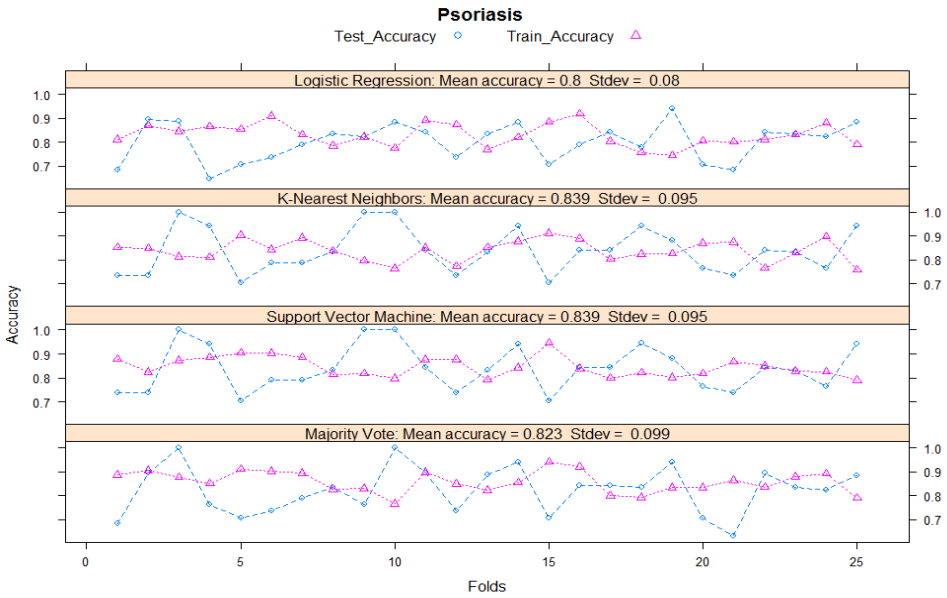


Figure 21: Psoriasis.

3.4 Psoriasis$_{RNAseq}$

The repeated nested cross validation, gave us that the best hyperparameters for Psoriasis RNAseq dataset are:

Psoriasis RNAseq: Hyperparameters For Classifiers			
SVM	Kernel = Linear	Cost = 0.001	
KNN	Metric = Manhattan	N Neighbors = 1	Weights = Uniform
Logistic Regression	Penalty = l1	Cost = 0.1	

From the confusions matrices below, illustrates that all the classifiers that tested estimated that they have the same accuracy fdr and f1. Our estimation is that we will predict the new labels with 98.88% accuracy. Also, it is more probably that this 1.12% error will be type error II. With nested cv our estimation is conserved and this confirmed by the results in Kaggle with 0 errors.

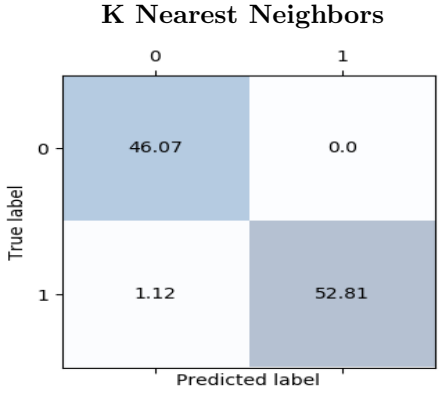


Figure 22: $Accuracy = 98.88\%$, $FDR = 0\%$, $F1 = 98.8\%$.

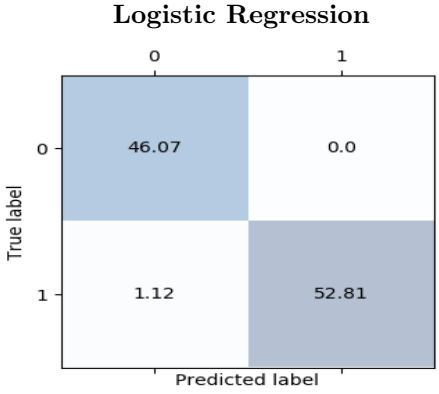


Figure 23: $Accuracy = 98.88\%$, $FDR = 0\%$, $F1 = 98.8\%$.

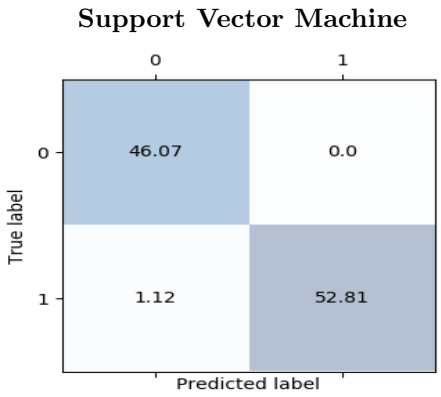


Figure 24: $Accuracy = 98.88\%$, $FDR = 0\%$, $F1 = 98.8\%$.

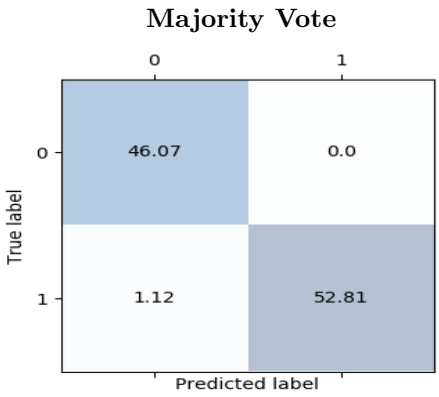


Figure 25: $Accuracy = 98.88\%$, $FDR = 0\%$, $F1 = 98.8\%$.

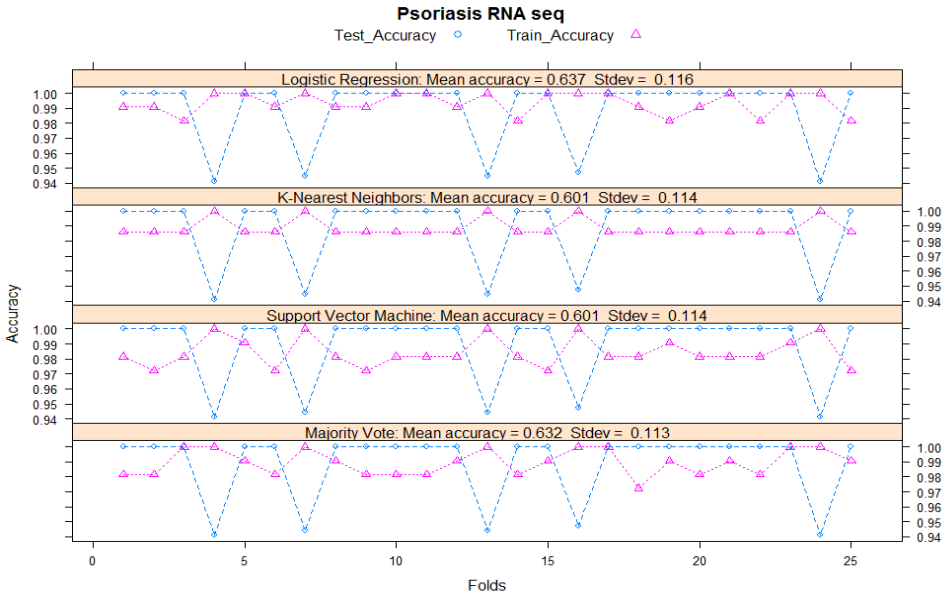


Figure 26: Psoriasis RNA-seq

4 Conclusion

When we deal with medical data, (binary classes: control vs disease), we must be very careful which classifier we must choose. From some point of view, we must treat our results like a hypothesis testing. If we estimated that our classifier can not predict with over 95% accuracy the results, we must ready to make a decision of which type error, we prefer to increase. In medical cases, we must keep the type error II low. A type II error is the failure to reject a false null hypothesis. An example in our case, is that we fail to predict that a person-sample has Lupus. In constant, type I error is the incorrect rejection of a true null hypothesis. An example of type I errors include a test that shows a patient to have a psoriasis when in fact the patient does not have the disease. That kind of error is not so serious as type error II. So in case of our dataset, I will prefer a classifier that minimize the type error II and have the best accuracy.

5 Bibliography

References

[1] Friedman, Tibshirani, Hastie *Elements of Statistical Learning*

[2] Bishop *Pattern Recognition And Machine Learning*

[3] Sebastian Raschka *Python Machine Learning*

[4] Statnikov, Alexander Aliferis, Constantin F. Tsamardinos, Ioannis Hardin, Douglas Levy, Shawn *A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis.*

[5] S. Varma and R. Simon *Bias in Error Estimation When Using Cross-validation for Model Selection. BMC bioinformatics, 7(1):91, 2006*

[6] Mukherjee,S. (2003) *Classifying Microarray Data Using Support Vector Machines, Understanding And Using Microarray Analysis Techniques: A Practical Guide.Kluwer Academic Publishers, Boston, MA.*

[7] Kohavi R. *A Study of Cross-Validation and Bootstrap for Ac- curacy Estimation and Model Selection, IJCAI, 1995.*

[8] Statnikov, Alexander Aliferis, Constantin F. Tsamardinos, Ioannis *Methods for multi-category cancer diagnosis from gene expression data: A comprehensive evaluation to inform decision support system development.Studies in Health Technology and Informatics. 2004. 107 p.813-817*

[9] Powers, David M W *"Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness Correlation" (PDF). Journal of Machine Learning Technologies. 2011. 2 (1): 37–63.*

6 Appendix

File: Repeated Nested Tunning

```
# -*- coding: utf-8 -*-
"""
Created on May 2017
Project: Methods in Bioinformatics

@author: Dimitris Kyriakis
"""

#============================================================================#
#================================ LIBRARIES =================================#
#============================================================================#

print("\n\n##################\nLoadinLg Libraries")
## MAIN ##
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

## PIPELINE ##
from sklearn.pipeline import Pipeline
from sklearn.cross_validation import StratifiedKFold
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler

## FEATURE ##
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import Lasso
from sklearn.feature_selection import VarianceThreshold

## CLASSIFIERS ##
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.feature_selection import RFE
from sklearn.feature_selection import SelectPercentile,f_classif
transform = SelectPercentile(f_classif)
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
##############################################################################

#============================================================================#
#============================ CHOOSE DATA SET ===============================#
#============================================================================#

Working_Dir = "C:/Users/Eddie/Desktop/Master/2nd_Semester/Methods_in_Bioinformatics/Tsamard/Exercise/"

Choose_Data_Set = input("""Choose_Data_Set:\n\t
         1.Lupus\n\t
         2.Psoriasis\n\t
         3.Autism\n\t
         4.Psoriasis_RNAseq\n\t
 (1/2/3/4): """)
#Choose_Data_Set = "3"

if Choose_Data_Set =="1":
    Train = Working_Dir + "Lupus/SRP062966Give.csv"
    Test = Working_Dir + "Lupus/SRP062966Validation.csv"
    Case= "Lupus"
    control = "control"
elif Choose_Data_Set == "2":
    Train = Working_Dir + "Psoriasis/GDS4602Give.csv"
    Test = Working_Dir + "Psoriasis/GDS4602Validation.csv"
    Case= "Psoriasis"
    control = "healthy"
elif Choose_Data_Set == "3":
    Train = Working_Dir + "Autism/GDS4431Give.csv"
    Test = Working_Dir + "Autism/GDS4431Validation.csv"
    Case= "Autism"
    control = "control"
else:
    Train = Working_Dir + "Psoriasis_RNAseq/SRP035988Give.csv"
    Test = Working_Dir + "Psoriasis_RNAseq/SRP035988Validation.csv"
    Case= "Psoriasis_RNAseq"

print("\n\n##################\n"+Case + " Choosed\n")

Data = np.genfromtxt(Train, delimiter=',', dtype=str)
Test_Data_raw = np.genfromtxt(Test, delimiter=',', dtype=str)
Test_Data_val = Test_Data_raw[1:,1:].astype(np.float)
Test_Data_samples = Test_Data_raw[1:,0]
Samples_Name = Data[1:,0].reshape(Data.shape[0]-1,1)
```

```
Features = Data[0,1:]

Train_Data = (Data[1:,1:Data.shape[1]-1]).astype(np.float)

Labels_raw = Data[1:,Data.shape[1]-1]#.reshape(Data.shape[0]-1,1)
Labels_raw = Labels_raw != control
Labels = Labels_raw.astype(int)
count_control = sum(Labels == 0)
Min_Folds = min(count_control, Labels.shape[0] - count_control)

print("\n\n##################\n"+"Data Loaded\n")




################################################################################

#================================================================================#
#============================== Choose Classifier ===============================#
#================================================================================#
clas = input("Choose Classifier\n1. KNN\n2.Logistic Regression\n3. SVM\n((1/2/3) = ")

def Classifier_Ch(choice_cl,Min_F,thres,S,D):
    '''
    **Description:**\n
    Create a pipeline with stdscale, variance feature selection and classifier (user input).\n
    **Input:**\n
    - Classifier: 1 = KNN, 2 = Logistic Regression, 3. SVM\n
    - Min_f: Number of Folds\n
    - Variance threshold\n
    **Output:**\n
    - Pipeline
    '''
    ## Assign Scaler  ##
    scaler = StandardScaler()
#     scaler = MinMaxScaler()
    ## Assign Feature selection ##
#     selector = VarianceThreshold(threshold = 0.001)
    ## Assign Classifier ##
    if choice_cl == "1" :
        name = "KNN"
        clf = KNeighborsClassifier()
        param_grid = [{'clf__n_neighbors': list(range(1, 20,2)),'clf__metric':
            ['minkowski','euclidean','manhattan'] ,'clf__weights':['uniform','distance']}]
    elif choice_cl == "2" :
        name = "LG"
        clf = LogisticRegression()
        scaler = MinMaxScaler()
        param_grid = [{'clf__penalty': ['l1','l2'],'clf__C': [0.0001, 0.001, 0.01, 0.1, 10, 100]}]
    elif choice_cl == "3":
        clf1 = KNeighborsClassifier(n_neighbors=19,metric='manhattan',weights='uniform')
        clf2 = SVC(C =  0.0001, kernel = 'linear')
        clf3 = LogisticRegression(C = 0.01, penalty = 'l2')
        pipe1 = Pipeline([('std', StandardScaler()),('clf1', clf1)])
        pipe2 = Pipeline([('std', StandardScaler()),('clf2', clf2)])
        pipe3 = Pipeline([('std', StandardScaler()),('clf3', clf3)])
        clf = VotingClassifier(estimators=[('clf1',pipe1),('clf2', pipe2),
            ('clf3',pipe3)],voting='soft')
    else:
        name = "SVM"
        clf = SVC()
        param_grid = [{'clf__kernel': ['linear'],'clf__C': [0.0001, 0.001, 0.01, 0.1, 10,
            100]},{'clf__kernel': ['poly'],'clf__C': [0.0001, 0.001, 0.01, 0.1, 10,
            100],'clf__degree' :[2,3]},
        {'clf__kernel': ['rbf'],'clf__C': [0.0001, 0.001, 0.01, 0.1, 10, 100],'clf__gamma': [0.001,
            0.0001]}]


    ## Create Pipeline ##
    pipe = Pipeline([('std', scaler),('clf', clf)])
    gcv = GridSearchCV(estimator=pipe,param_grid=param_grid,scoring='accuracy',cv=Min_F)
    return gcv,name

if clas == "1":
    name = "KNN"
elif clas == "2":
    name = "LG"
else:
    name = "SVM"


################################################################################

#VotingClassifier(estimators=[('clf1',pipe1),('clf2', pipe2), ('clf3',pipe3)],voting='soft')



#================================================================================#
```

```
#=================== FEATURE SELECTION ======================================#
#============================================================================#
thres = 0.01
in_out = "IN"


#============================================================================#
#================================ MAIN CODE =================================#
#============================================================================#

if Min_Folds >=10:
    Min_Folds = 10
Min_Folds =5

mat = np.zeros((2,2))
mean_acc = 0
Best_Param = {}
lista_acc =[]
### REPEATED NESTED ###
output = open("Results/"+name+"_"+Case+".txt","w")
for  i in [1,2,3,4,5]:
    ## NESTED CROSS VALIDATION ##
    kfold = StratifiedKFold(y=Labels, n_folds= Min_Folds, shuffle=True)


    svm_list = []
    knn_list =[]
    lg_list = []

    for train_idx, test_idx in kfold:
        ## Assign ##
        inner_Train = Train_Data[train_idx]
        S = inner_Train.shape[0]
        D = inner_Train.shape[1]
        inner_Test = Train_Data[test_idx]
        inner_labels = Labels[train_idx]
        count_control_inner = sum(inner_labels == 0)
        Min_Folds_inner = min(count_control_inner, inner_labels.shape[0] - count_control_inner)
        if Min_Folds_inner >=10:
            Min_Folds_inner = 9
        ## Prepare ##

        gcv,name = Classifier_Ch(clas ,Min_Folds_inner,thres,S,D)
        ## CROSS VALIDATION ##
        gcv.fit(inner_Train,inner_labels)
        best_par = gcv.best_params_

        if str(best_par) in Best_Param.keys():
            Best_Param[str(best_par)] +=1
        else:
            Best_Param[str(best_par)] = 1
        ## Predict ##
        y_test = Labels[test_idx]
        y_pred = gcv.predict(inner_Test)

        ## Accuracy ##
        acc = accuracy_score(y_true=y_test, y_pred=y_pred)
        lista_acc.append(acc)
        output.write("\n"+ str(best_par)+' | inner ACC %.2f%% | outer ACC %.2f%%' % (gcv.best_score_
            * 100, acc * 100))
        mean_acc = mean_acc + acc
    print(i)


output.write(str(mean_acc/(5*Min_Folds)))
output.write(str(lista_acc))
output.write(str(Best_Param))
output.close()
###############################################################################









#============================================================================#
#================================== END =====================================#
#============================================================================#
#
### 1.  Assign Classifier ##
#if Choose_Data_Set =="1":
#    clf = SVC(C = 0.0001,  kernel = 'linear')
#elif Choose_Data_Set == "2":
#    clf = SVC(C = 0.0001 , kernel = 'linear')
#elif Choose_Data_Set == "3":
#    clf = SVC(C = 0.0001 , kernel = 'linear')
```

```
#else:
#    clf = SVC(C = 0.0001 , kernel = 'linear')
#
#
### 2.  Feature ##
#
#scaler = StandardScaler()
#selector = VarianceThreshold(threshold = 1)
#
## 3.  Prepare ##
#pipe = Pipeline([('std', scaler),('feat', selector),('clf', clf)])
pipe = Pipeline([('std', scaler),('clf', clf)])
#pipe.fit(Train_Data,Labels)
#
### 4.  Predict ##
#predictions  = pipe.predict(Test_Data_val)
#
## 5.  Write ##
Test_Data_samples = list(range(1,predictions.shape[0]+1))
predict = {'Id': Test_Data_samples, 'Predicted':list(predictions)}
predict = pd.DataFrame(predict)
predict.to_csv(Working_Dir + Case +"_predictions_Lasso_"+in_out+"_thres"+ str(thres) +".csv", index =
    False)
```

File: Ensemble- Plots

```
## Libraries ##
## PIPELINE ##
from sklearn.pipeline import Pipeline
from sklearn.cross_validation import StratifiedKFold
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler



## CLASSIFIERS ##
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.feature_selection import RFE
from sklearn.feature_selection import SelectPercentile,f_classif
transform = SelectPercentile(f_classif)
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.cross_validation import cross_val_score
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt


#============================================================================#
#============================ CHOOSE DATA SET ===============================#
#============================================================================#

Working_Dir = "C:/Users/Eddie/Desktop/Master/2nd_Semester/Methods_in_Bioinformatics/Tsamard/Exercise/"

Choose_Data_Set = input("""Choose_Data_Set:\n\t
         1.Lupus\n\t
         2.Psoriasis\n\t
         3.Autism\n\t
         4.Psoriasis_RNAseq\n\t
 (1/2/3/4): """)

#Choose_Data_Set = "2"

if Choose_Data_Set =="1":
    Train = Working_Dir + "Lupus/SRP062966Give.csv"
    Test = Working_Dir + "Lupus/SRP062966Validation.csv"
    Case= "Lupus"
    control = "control"
    ## Best hyperparameters ## CHECKED
    clf1 = LogisticRegression(penalty='l2',C= 10) #NAI
    clf2 = KNeighborsClassifier(metric='manhattan',n_neighbors=5) #NAI
    clf3 = SVC(kernel = 'linear', C = 0.0001,probability=True) #NAI
elif Choose_Data_Set == "2":
    Train = "Psoriasis/GDS4602Give.csv"
    Test = "Psoriasis/GDS4602Validation.csv"
    Case= "Psoriasis"
    control = "healthy"
    ## Best hyperparameters ##
    clf1 = LogisticRegression(penalty='l1',C= 100) #NAI
    clf2 = KNeighborsClassifier(metric='minkowski',n_neighbors=3) # NAI
    clf3 = SVC(kernel = 'linear', C = 0.0001,probability=True) #NAI

elif Choose_Data_Set == "3":
    Train = Working_Dir + "Autism/GDS4431Give.csv"
    Test = Working_Dir + "Autism/GDS4431Validation.csv"
    Case= "Autism"
    control = "control"
    ## Best hyperparameters ## CHECKED
    clf1 = LogisticRegression(penalty='l2',C= 0.001) #NAI
    clf2 = KNeighborsClassifier(metric='minkowski',n_neighbors=7) #NAI
    clf3 = SVC(kernel = 'linear', C = 0.0001,probability=True)  #NAI
else:
    Train = Working_Dir + "Psoriasis_RNAseq/SRP035988Give.csv"
    Test = Working_Dir + "Psoriasis_RNAseq/SRP035988Validation.csv"
    Case= "Psoriasis_RNAseq"
    control = "control"
    ## Best hyperparameters ##
    clf1 = LogisticRegression(penalty='l1',C= 0.1) #NAI
    clf2 = KNeighborsClassifier(metric='manhattan',n_neighbors=1) # NAI
    clf3 = SVC(kernel = 'linear', C = 0.0001,probability=True) #NAI


print("\n\n##################\n"+Case + " Choosed\n")

```

```
Data = np.genfromtxt(Train, delimiter=',', dtype=str)
Test_Data_raw = np.genfromtxt(Test, delimiter=',', dtype=str)
Test_Data_val = Test_Data_raw[1:,1:].astype(np.float) # VALUATION
Test_Data_samples = Test_Data_raw[1:,0]
Samples_Name = Data[1:,0].reshape(Data.shape[0]-1,1)
Features = Data[0,1:]

Train_Data = (Data[1:,1:Data.shape[1]-1]).astype(np.float) # TRAIN

Labels_raw = Data[1:,Data.shape[1]-1]
Labels_raw = Labels_raw != control
Labels = Labels_raw.astype(int) # LABELS
count_control = sum(Labels == 0)
Min_Folds = min(count_control, Labels.shape[0] - count_control)

print("\n\n##################\n"+"Data Loaded\n")



# Building the pipelines
pipe1 = Pipeline([('std', StandardScaler()),('clf1', clf1)])
pipe2 = Pipeline([('std', StandardScaler()),('clf2', clf2)])
pipe3 = Pipeline([('std', StandardScaler()),('clf3', clf3)])

#
#
# ENSEMBLE #
if Min_Folds >=10:
    Min_Folds = 10
Min_Folds =5

mat = np.zeros((2,2))
mat_lg = np.zeros((2,2))
mat_knn = np.zeros((2,2))
mat_svm = np.zeros((2,2))
mean_acc = 0

clf_labels = ['Logistic Regression', 'KNN', 'SVM','Majority Voting']
Dict_acc = {'Logistic Regression':[], 'KNN':[], 'SVM':[],'Majority Voting': []}
Dict_val = {'Logistic Regression':[], 'KNN':[], 'SVM':[],'Majority Voting': []}
Dict_std = {'Logistic Regression':[], 'KNN':[], 'SVM':[],'Majority Voting': []}
### REPEATED NESTED ###
for  i in [1,2,3,4,5]:
    ## NESTED CROSS VALIDATION ##
    kfold = StratifiedKFold(y=Labels, n_folds= Min_Folds, shuffle=True)

    for train_idx, test_idx in kfold:
        ## Assign ##
        inner_Train = Train_Data[train_idx]
        S = inner_Train.shape[0]
        D = inner_Train.shape[1]
        inner_Test = Train_Data[test_idx]
        inner_labels = Labels[train_idx]
        count_control_inner = sum(inner_labels == 0)
        Min_Folds_inner = min(count_control_inner, inner_labels.shape[0] - count_control_inner)
        if Min_Folds_inner >=10:
            Min_Folds_inner = 9

        mv_clf =  VotingClassifier(estimators=[('clf1',pipe1),('clf2', pipe2),
            ('clf3',pipe3)],voting='soft')
        all_clf = [pipe1, pipe2, pipe3, mv_clf]

        ## Cross Validation ##
        for clf, label in zip(all_clf, clf_labels):
            scores =
                cross_val_score(estimator=clf,X=inner_Train,y=inner_labels,cv=Min_Folds_inner,scoring='roc_auc',)
            print("Accuracy: %0.2f (+/- %0.2f) [%s]"% (scores.mean(), scores.std(), label))
            Dict_val[label].append(scores.mean())
            Dict_std[label].append(scores.std())
        ## EMSEMBLE ##
        eAlg = VotingClassifier(estimators=[('clf1',pipe1),('clf2', pipe2),
            ('clf3',pipe3)],voting='soft')
        eAlg.fit(inner_Train,inner_labels)
        y_test = Labels[test_idx]
        pred_ens = eAlg.predict(inner_Test)
        confmat = confusion_matrix(y_true=y_test, y_pred=pred_ens)
        mat += confmat
        acc = accuracy_score(y_true=y_test, y_pred=pred_ens)
        Dict_acc['Majority Voting'].append(acc)

        ### LG ###
        pipe1.fit(inner_Train,inner_labels)
        pred_lg = pipe1.predict(inner_Test)
        confmat = confusion_matrix(y_true=y_test, y_pred=pred_lg)
        mat_lg += confmat
        acc = accuracy_score(y_true=y_test, y_pred=pred_lg)
        Dict_acc['Logistic Regression'].append(acc)

```

```
        ### KNN ###
        pipe2.fit(inner_Train,inner_labels)
        pred_knn = pipe2.predict(inner_Test)
        confmat = confusion_matrix(y_true=y_test, y_pred=pred_knn)
        mat_knn += confmat
        acc = accuracy_score(y_true=y_test, y_pred=pred_knn)
        Dict_acc['KNN'].append(acc)

        ### SVM ###
        pipe3.fit(inner_Train,inner_labels)
        pred_svm = pipe3.predict(inner_Test)
        confmat = confusion_matrix(y_true=y_test, y_pred=pred_svm)
        mat_svm += confmat
        Dict_acc['SVM'].append(acc)



Store = Working_Dir+"Tables_PNG/"

## KNN MEAN MATRIX ##
mat_knn_mean= np.round(mat_knn/sum(sum(mat_knn))*100,2)
acc = mat_knn_mean[0,0]+mat_knn_mean[1,1]
fdr = mat_knn_mean[0,1]/(mat_knn_mean[0,1]+mat_knn_mean[0,0])
f1 = (2*mat_knn_mean[0,0])/((2*mat_knn_mean[0,0])+mat_knn_mean[1,0]+mat_knn_mean[0,1])
print("KNN")
print("Accuracy = "+str(acc)+"%")
print("FDR = "+str(round(fdr*100,2))+"%")
print("F1 = "+str(round(f1*100,2))+"%")


fig, cx = plt.subplots(figsize=(3.5, 3.5))
cx.matshow(mat_knn_mean, cmap=plt.cm.Blues, alpha=0.3)
for i in range(mat_knn_mean.shape[0]):
    for j in range(mat_knn_mean.shape[1]):
        cx.text(x=j, y=i,s=mat_knn_mean[i, j],va='center', ha='center')

plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.savefig(Store+Case+'_KNN_Mean.png')



## LG MEAN MATRIX ##
mat_lg_mean= np.round(mat_lg/sum(sum(mat_lg))*100,2)
acc = mat_lg_mean[0,0]+mat_lg_mean[1,1]
fdr = mat_knn_mean[0,1]/(mat_knn_mean[0,1]+mat_knn_mean[0,0])
f1 = (2*mat_lg_mean[0,0])/((2*mat_lg_mean[0,0])+mat_lg_mean[1,0]+mat_lg_mean[0,1])
print("LG")
print("Accuracy = "+str(acc)+"%")
print("FDR = "+str(round(fdr*100,2))+"%")
print("F1 = "+str(round(f1*100,2))+"%")

fig, bx = plt.subplots(figsize=(3.5, 3.5))
bx.matshow(mat_lg_mean, cmap=plt.cm.Blues, alpha=0.3)
for i in range(mat_lg_mean.shape[0]):
    for j in range(mat_lg_mean.shape[1]):
        bx.text(x=j, y=i,s=mat_lg_mean[i, j],va='center', ha='center')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.savefig(Store+Case+'_LG_Mean.png')




## SVM MEAN MATRIX ##
mat_svm_mean= np.round(mat_svm/sum(sum(mat_svm))*100,2)
acc = mat_svm_mean[0,0]+mat_svm_mean[1,1]
fdr = mat_knn_mean[0,1]/(mat_knn_mean[0,1]+mat_knn_mean[0,0])
f1 = (2*mat_svm_mean[0,0])/((2*mat_svm_mean[0,0])+mat_svm_mean[1,0]+mat_svm_mean[0,1])
print("SVM")
print("Accuracy = "+str(acc)+"%")
print("FDR = "+str(round(fdr*100,2))+"%")
print("F1 = "+str(round(f1*100,2))+"%")

fig, dx = plt.subplots(figsize=(3.5, 3.5))
dx.matshow(mat_svm_mean, cmap=plt.cm.Blues, alpha=0.3)
for i in range(mat_svm_mean.shape[0]):
    for j in range(mat_svm_mean.shape[1]):
        dx.text(x=j, y=i,s=mat_svm_mean[i, j],va='center', ha='center')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.savefig(Store+Case+'_SVM_Mean.png')



## MEAN MATRIX ##
mat3= np.round(mat/sum(sum(mat))*100,2)
```

```
acc = mat3[0,0]+mat3[1,1]
fdr = mat_knn_mean[0,1]/(mat_knn_mean[0,1]+mat_knn_mean[0,0])
f1 = (2*mat3[0,0])/((2*mat3[0,0])+mat3[1,0]+mat3[0,1])
print("Majority")
print("Accuracy = "+str(acc)+"%")
print("FDR = "+str(round(fdr*100,2))+"%")
print("F1 = "+str(round(f1*100,2))+"%")



fig, ax = plt.subplots(figsize=(3.5, 3.5))
ax.matshow(mat3, cmap=plt.cm.Blues, alpha=0.3)
for i in range(mat3.shape[0]):
    for j in range(mat3.shape[1]):
        ax.text(x=j, y=i,s=mat3[i, j],va='center', ha='center')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.savefig(Store+Case+'_Mean.png')

print("Test")
print(Dict_acc)
print("\n\n\n")
print("TRAIN")
print(Dict_val)
print("\n\n\n")
print(Dict_std)

if Case =="Lupus":
    final_pipe = pipe3
elif Case =="Psoriasis":
    Test_Data_samples=list(range(1,Test_Data_val.shape[0]+1))
    final_pipe = pipe2
    final_pipe = VotingClassifier(estimators=[('clf1',pipe1),('clf2', pipe2),
        ('clf3',pipe3)],voting='soft')
elif Case == "Psoriasis_RNAseq":
    Test_Data_samples=list(range(1,Test_Data_val.shape[0]+1))
    final_pipe = VotingClassifier(estimators=[('clf1',pipe1),('clf2', pipe2),
        ('clf3',pipe3)],voting='soft')
elif Case == "Autism":
    Test_Data_samples =list(range(1,Test_Data_val.shape[0]+1))
    final_pipe = pipe1

final_pipe.fit(Train_Data,Labels)
pred = final_pipe.predict(Test_Data_val)
predict = {'Id': Test_Data_samples, 'Predicted':list(pred)}
prediction = pd.DataFrame(predict)
prediction.to_csv(Case+'_predict_Ensemble.csv', index = False)
```

File:Extra Figures

```
################################## LUPUS ##########################################

LG <- c(0.83333333333333337, 0.83333333333333337, 0.83333333333333337, 0.83333333333333337,
    0.90909090909090906, 0.83333333333333337, 0.83333333333333337, 0.83333333333333337,
    0.83333333333333337, 0.90909090909090906, 0.83333333333333337, 0.83333333333333337,
    0.83333333333333337, 0.83333333333333337, 0.90909090909090906, 0.83333333333333337,
    0.83333333333333337, 0.83333333333333337, 0.83333333333333337, 0.90909090909090906,
    0.83333333333333337, 0.83333333333333337, 0.83333333333333337, 0.83333333333333337,
    0.90909090909090906)
KNN <- c(0.75, 1.0, 1.0, 0.83333333333333337, 1.0, 1.0, 1.0, 1.0, 0.75, 0.72727272727272729, 1.0,
    0.91666666666666663, 0.83333333333333337, 0.91666666666666663, 0.90909090909090906, 1.0,
    0.91666666666666663, 0.83333333333333337, 0.91666666666666663, 0.90909090909090906,
    0.83333333333333337, 0.91666666666666663, 0.83333333333333337, 1.0, 0.81818181818181823)
SVM <- c(0.83333333333333337, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.91666666666666663, 0.91666666666666663,
    1.0, 1.0, 0.91666666666666663, 1.0, 1.0, 1.0, 1.0, 1.0, 0.91666666666666663, 1.0, 1.0,
    0.91666666666666663, 1.0, 1.0, 1.0, 1.0)
MV <- c(0.83333333333333337, 1.0, 0.91666666666666663, 1.0, 1.0, 1.0, 0.91666666666666663,
    0.91666666666666663, 0.83333333333333337, 1.0, 1.0, 0.91666666666666663, 0.83333333333333337,
    0.91666666666666663, 1.0, 1.0, 1.0, 0.91666666666666663, 0.91666666666666663, 1.0,
    0.83333333333333337, 1.0, 1.0, 0.91666666666666663, 1.0)


lg_title = paste(paste('Logistic Regression: Mean accuracy =',round(mean(LG),3)),paste(" Stdev =
    ",round(sd(LG),3)))
knn_title = paste(paste('K-Nearest Neighbors: Mean accuracy =',round(mean(KNN),3)),paste(" Stdev =
    ",round(sd(KNN),3)))
svm_title = paste(paste('Support Vector Machine: Mean accuracy =',round(mean(SVM),3)),paste(" Stdev =
    ",round(sd(SVM),3)))
mv_title = paste(paste('Majority Vote: Mean accuracy =',round(mean(MV),3)),paste(" Stdev =
    ",round(sd(MV),3)))

title <- rep(c(lg_title,knn_title,svm_title,mv_title),each=25)

labels <-rep(c('KNN','SVM','Majority Voting'),each =25)
nums = cbind(LG,KNN,SVM,MV)

as <- melt(nums)
sa <- data.frame(as,title)
sa
xyplot (value ~ Var1 | title, groups=Var2, data=sa, type="o",
        layout=c(1, 4), as.table=T, xlab="Folds", ylab="Accuracy",main = "Lupus")


############################ AUTISM ##############################
LG <- c(0.53333333333333333, 0.8666666666666667, 0.73333333333333328, 0.46666666666666667,
    0.7142857142857143, 0.46666666666666667, 0.8666666666666667, 0.73333333333333328,
    0.66666666666666663, 0.42857142857142855, 0.53333333333333333, 0.66666666666666663,
    0.59999999999999998, 0.53333333333333333, 0.6428571428571429, 0.73333333333333328,
    0.66666666666666663, 0.46666666666666667, 0.46666666666666667, 0.5714285714285714,
    0.66666666666666663, 0.80000000000000004, 0.66666666666666663, 0.59999999999999998,
    0.5714285714285714)
KNN <- c(0.59999999999999998, 0.73333333333333328, 0.73333333333333328, 0.59999999999999998,
    0.6428571428571429, 0.73333333333333328, 0.73333333333333328, 0.59999999999999998,
    0.53333333333333333, 0.5, 0.59999999999999998, 0.73333333333333328, 0.59999999999999998,
    0.46666666666666667, 0.7142857142857143, 0.66666666666666663, 0.73333333333333328,
    0.53333333333333333, 0.66666666666666663, 0.5714285714285714, 0.53333333333333333,
    0.73333333333333328, 0.59999999999999998, 0.59999999999999998, 0.7142857142857143)
SVM <- c(0.73333333333333328, 0.80000000000000004, 0.66666666666666663, 0.53333333333333333,
    0.7857142857142857, 0.53333333333333333, 0.80000000000000004, 0.73333333333333328,
    0.66666666666666663, 0.42857142857142855, 0.53333333333333333, 0.59999999999999998,
    0.66666666666666663, 0.46666666666666667, 0.5714285714285714, 0.66666666666666663,
    0.66666666666666663, 0.53333333333333333, 0.53333333333333333, 0.5, 0.66666666666666663,
    0.73333333333333328, 0.66666666666666663, 0.59999999999999998, 0.5714285714285714)
MV <- c(0.59999999999999998, 0.80000000000000004, 0.73333333333333328, 0.53333333333333333,
    0.7142857142857143, 0.53333333333333333, 0.8666666666666667, 0.59999999999999998,
    0.53333333333333333, 0.5, 0.59999999999999998, 0.66666666666666663, 0.59999999999999998,
    0.53333333333333333, 0.6428571428571429, 0.73333333333333328, 0.80000000000000004,
    0.53333333333333333, 0.46666666666666667, 0.5714285714285714, 0.66666666666666663,
    0.80000000000000004, 0.59999999999999998, 0.73333333333333328, 0.6428571428571429)

lg_title = paste(paste('Logistic Regression: Mean accuracy =',round(mean(LG),3)),paste(" Stdev =
    ",round(sd(LG),3)))
knn_title = paste(paste('K-Nearest Neighbors: Mean accuracy =',round(mean(KNN),3)),paste(" Stdev =
    ",round(sd(KNN),3)))
svm_title = paste(paste('Support Vector Machine: Mean accuracy =',round(mean(SVM),3)),paste(" Stdev =
    ",round(sd(SVM),3)))
mv_title = paste(paste('Majority Vote: Mean accuracy =',round(mean(MV),3)),paste(" Stdev =
    ",round(sd(MV),3)))

title <- rep(c(lg_title,knn_title,svm_title,mv_title),each=25)

labels <-rep(c('KNN','SVM','Majority Voting'),each =25)
nums = cbind(LG,KNN,SVM,MV)

as <- melt(nums)
sa <- data.frame(as,title)
sa
```

```
xyplot (value ~ Var1 | title, groups=Var2, data=sa, type="o",
        layout=c(1, 4), as.table=T, xlab="Folds", ylab="Accuracy",main = "Autism")


################################ PSORIASIS ###################################

LG <- c(0.78947368421052633, 0.78947368421052633, 0.83333333333333337, 0.88235294117647056,
     0.82352941176470584, 0.78947368421052633, 0.84210526315789469, 0.88888888888888884,
     0.82352941176470584, 0.82352941176470584, 0.89473684210526316, 0.73684210526315785,
     0.66666666666666663, 0.82352941176470584, 0.88235294117647056, 0.73684210526315785,
     0.84210526315789469, 0.88888888888888884, 0.88235294117647056, 0.76470588235294112,
     0.89473684210526316, 0.84210526315789469, 0.88888888888888884, 0.82352941176470584,
     0.82352941176470584)
KNN <- c(0.89473684210526316, 0.68421052631578949, 0.94444444444444442, 0.76470588235294112,
     0.88235294117647056, 0.84210526315789469, 0.84210526315789469, 0.88888888888888884,
     0.76470588235294112, 0.70588235294117652, 0.78947368421052633, 0.78947368421052633,
     0.72222222222222221, 0.82352941176470584, 0.94117647058823528, 0.78947368421052633,
     0.84210526315789469, 0.88888888888888884, 0.94117647058823528, 0.82352941176470584,
     0.89473684210526316, 0.84210526315789469, 0.83333333333333337, 0.76470588235294112,
     0.82352941176470584)
SVM <- c(0.84210526315789469, 0.78947368421052633, 0.83333333333333337, 0.82352941176470584,
     0.76470588235294112, 0.78947368421052633, 0.89473684210526316, 0.88888888888888884,
     0.82352941176470584, 0.82352941176470584, 0.78947368421052633, 0.89473684210526316,
     0.77777777777777779, 0.82352941176470584, 0.88235294117647056, 0.73684210526315785,
     0.78947368421052633, 0.88888888888888884, 0.94117647058823528, 0.76470588235294112,
     0.89473684210526316, 0.84210526315789469, 0.88888888888888884, 0.76470588235294112,
     0.76470588235294112)
MV<- c(0.84210526315789469, 0.73684210526315785, 0.88888888888888884, 0.82352941176470584,
     0.82352941176470584, 0.84210526315789469, 0.84210526315789469, 0.83333333333333337,
     0.82352941176470584, 0.82352941176470584, 0.84210526315789469, 0.78947368421052633,
     0.72222222222222221, 0.82352941176470584, 0.88235294117647056, 0.78947368421052633,
     0.84210526315789469, 0.88888888888888884, 0.94117647058823528, 0.82352941176470584,
     0.89473684210526316, 0.89473684210526316, 0.88888888888888884, 0.82352941176470584,
     0.82352941176470584)


LG<-c(0.78947368421052633, 0.84210526315789469, 0.61111111111111116, 0.88235294117647056,
     0.82352941176470584, 0.84210526315789469, 0.78947368421052633, 0.72222222222222221,
     0.88235294117647056, 0.76470588235294112, 0.57894736842105265, 0.68421052631578949,
     0.77777777777777779, 0.88235294117647056, 0.88235294117647056, 0.73684210526315785,
     0.68421052631578949, 0.94444444444444442, 0.82352941176470584, 0.88235294117647056,
     0.84210526315789469, 0.68421052631578949, 0.77777777777777779, 0.88235294117647056,
     0.82352941176470584)
KNN<-c(0.84210526315789469, 0.73684210526315785, 0.66666666666666663, 0.88235294117647056,
     0.94117647058823528, 0.84210526315789469, 0.78947368421052633, 0.83333333333333337,
     0.82352941176470584, 0.88235294117647056, 0.78947368421052633, 0.73684210526315785,
     0.83333333333333337, 0.76470588235294112, 0.94117647058823528, 0.78947368421052633,
     0.73684210526315785, 1.0, 0.70588235294117652, 1.0, 0.94736842105263153, 0.84210526315789469,
     0.77777777777777779, 0.88235294117647056, 0.76470588235294112)
SVM<-c(0.78947368421052633, 0.84210526315789469, 0.66666666666666663, 0.82352941176470584,
     0.94117647058823528, 0.78947368421052633, 0.73684210526315785, 0.88888888888888884,
     0.88235294117647056, 0.82352941176470584, 0.63157894736842102, 0.68421052631578949,
     0.83333333333333337, 0.88235294117647056, 0.94117647058823528, 0.78947368421052633,
     0.73684210526315785, 1.0, 0.82352941176470584, 0.88235294117647056, 0.84210526315789469,
     0.78947368421052633, 0.83333333333333337, 0.88235294117647056, 0.82352941176470584)
MV <-c(0.84210526315789469, 0.78947368421052633, 0.66666666666666663, 0.88235294117647056,
     0.94117647058823528, 0.84210526315789469, 0.78947368421052633, 0.83333333333333337,
     0.88235294117647056, 0.88235294117647056, 0.68421052631578949, 0.73684210526315785,
     0.83333333333333337, 0.88235294117647056, 0.94117647058823528, 0.78947368421052633,
     0.73684210526315785, 1.0, 0.76470588235294112, 1.0, 0.94736842105263153, 0.78947368421052633,
     0.83333333333333337, 0.88235294117647056, 0.76470588235294112)


lg_title = paste(paste('Logistic Regression: Mean accuracy =',round(mean(LG),3)),paste(" Stdev =
     ",round(sd(LG),3)))
knn_title = paste(paste('K-Nearest Neighbors: Mean accuracy =',round(mean(KNN),3)),paste(" Stdev =
     ",round(sd(KNN),3)))
svm_title = paste(paste('Support Vector Machine: Mean accuracy =',round(mean(SVM),3)),paste(" Stdev =
     ",round(sd(SVM),3)))
mv_title = paste(paste('Majority Vote: Mean accuracy =',round(mean(MV),3)),paste(" Stdev =
     ",round(sd(MV),3)))

title <- rep(c(lg_title,knn_title,svm_title,mv_title),each=25)

labels <-rep(c('KNN','SVM','Majority Voting'),each =25)
nums = cbind(LG,KNN,SVM,MV)

as <- melt(nums)
sa <- data.frame(as,title)
sa
xyplot (value ~ Var1 | title, groups=Var2, data=sa, type="o",
        layout=c(1, 4), as.table=T, xlab="Folds", ylab="Accuracy",main = "Psoriasis")


################################ PSORIASIS RNA seq ###################################


LG<-c(1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0,
```

```
    1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 1.0,
    0.94117647058823528)
KNN<-c(1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0,
    1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 1.0,
    0.94117647058823528)
SVM<-c(1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0,
    1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 1.0,
    0.94117647058823528)
MV<-c(1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0,
    1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 0.94117647058823528, 1.0, 1.0, 1.0, 1.0, 1.0,
    0.94117647058823528)


lg_title = paste(paste('Logistic Regression: Mean accuracy =',round(mean(LG),3)),paste(" Stdev =
    ",round(sd(LG),3)))
knn_title = paste(paste('K-Nearest Neighbors: Mean accuracy =',round(mean(KNN),3)),paste(" Stdev =
    ",round(sd(KNN),3)))
svm_title = paste(paste('Support Vector Machine: Mean accuracy =',round(mean(SVM),3)),paste(" Stdev =
    ",round(sd(SVM),3)))
mv_title = paste(paste('Majority Vote: Mean accuracy =',round(mean(MV),3)),paste(" Stdev =
    ",round(sd(MV),3)))

title <- rep(c(lg_title,knn_title,svm_title,mv_title),each=25)

labels <-rep(c('KNN','SVM','Majority Voting'),each =25)
nums = cbind(LG,KNN,SVM,MV)
as <- melt(nums)
sa <- data.frame(as,title)
xyplot (value ~ Var1 | title, groups=Var2, data=sa, type="o",
        layout=c(1, 4), as.table=T, xlab="Folds", ylab="Accuracy",main = "Psoriasis RNA seq")
```