Feature Selection

Harshil Shah
Computer Science
Colorado State University

November 17^{th}, 2015

Contents

---

1 Introduction

In this report, various feature selection methods are explored on two datasets - (i) Arcene Dataset & (ii) Leukemia Dataset. Python implementations regarding each approaches are appended in the end of the document as appendix section. Each shown implementation in appendix, depicts only main functionality of approaches. For full implementation kindly find enclosures.

2 Filter Methods

2.1 Golub Score

Golub score depicts Pearson Correlation of a feature with respect to the output. For given features $i = 1, 2 .. n$, Golub Score associated with each features can be calculated as follows.

$$GolubScore_i = \frac{|\mu_i^{(+)} - \mu_i^{(-)}|}{\sigma_i^{(+)} + \sigma_i^{(-)}}$$

Where:
$\mu_i^{(+)}$ = the average of feature i in the positive examples
$\sigma_i^{(+)}$ = the standard deviation of feature i in the positive examples
$\mu_i^{(-)}$ = the average of feature i in the negative examples
$\sigma_i^{(-)}$ = the standard deviation of feature i in the negative examples
$i = 1, 2, 3, .. n$

Implementation of Golub Score is shown in appendix B.1. Golub function returns two arrays in order for it to work with scikit learn's inbuilt functionalities.

3 Embedded Methods

3.1 Non-zero Weight Vector Coefficients Using L1-SVM

Number of features in each datasets are huge, but only a few are actually meaningful and participating in deciding output. L1-SVM gives sparse solution and can be used to select relevant features set. Implementation regarding selection is shown in appendix C.1. Implementation scans classifier attribute - coef array - to count total number of features having weights.

Dataset	Iterations Used	Average Number Of Non-Zero Weight Vector Coefficients	Range
Arcene	15	837	765.0 To 919.0
Leukemia	15	40	36.0 To 45.0

Table 1: Average Number Of Features Having Non Zero Weights

According to feature selection performed by L1-SVM, average number of features, which are having non-zero weights, are shown in above table 1. Range suggests span of number of possible features.

3.2 Accuracy Based On Different Approaches To Select Features And Classifiers

Different combinations of selector and classifier are used to reason accuracy based on different selectors for datasets. Implementations regarding each approaches are shown appendix - C.2, C.3 & C.4. Below table 2 shows accuracy generated by each approach on both datasets. To calculate accuracy cross-validation is used and pipeline functionality is used where all features are not taken for training the algorithm.

L1-SVM contains a linear summation of the slack variables while the L2-SVM contains a summation of the squared squared slack variables. Variance in such objectives gives variance in accuracy too.

Functional Requirements	Arcene Dataset		Leukemia Dataset	
	Iterations Used	Average Accuracy	Iterations Used	Average Accuracy
L1 SVM On All Feature	5	0.78323	5	0.96971
L2 SVM On Feature Selected By L1 SVM	3	0.88693	3	0.90857
L2 SVM On All Features	1	0.89919	1	0.81238
L2 SVM That Uses RFECV-L2 SVM To Select Features	1	0.90971	1	0.89285

Table 2: Comparison Of Approaches

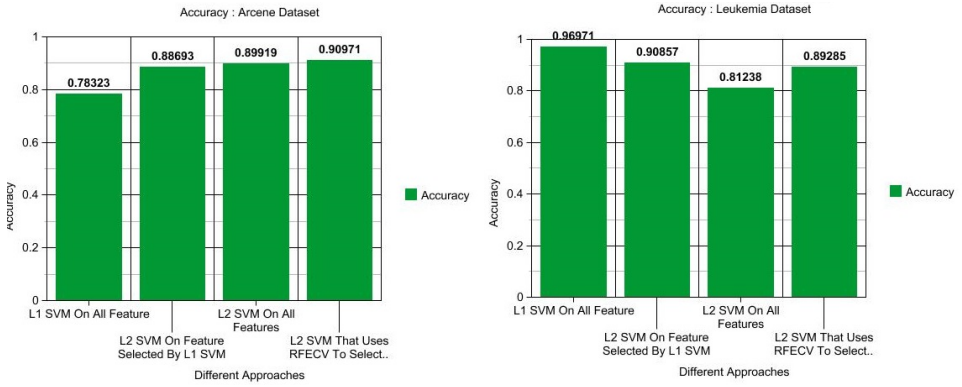

Figure 1: Accuracy Of Different Approaches On Datasets

Above bar graph - 1 - shows that for Arcene dataset L2 SVM works well than L1 does. But it is opposite in Leukemia dataset. Properties of algorithms satisfy above argument. Furthermore, features selected by RFECV always gives higher accuracy. Also, graph suggests that, For Leukemia dataset, L1 ends up removing relevant features too which should not be remove.

3.3 Score Matrix Created By Modified L1 SVM

Another approach to select feature is variance of L1-SVM. Implementation is shown in appendix C.6.

In this approach, training dataset is created by choosing random 80% of example from datasets. Training dataset then will be further divided into sub samples. Initially, feature sized array(score of feature) will be initialized to zero values. For each sub samples, L1-SVM will be train independently. For each associated features' non-zero weight in each L1-SVM(trained on sub samples), features' associated score will be incremented. Final score matrix will be deciding matrix to choose relevance of each features based on score. This approach will remove bias of each feature on output.

Below shown table 3 shows average number of features having non-zero scores in score matrix. Golub scores and score derived via this approach will be further used in upcoming section to select features.

Dataset Used	Iterations	Number Of Sub Samples	Size Of Each Sub Sample (Number Of Examples)	Average Number Of Non-Zero Weight Vector Coefficients	Range
Arcene	10	8	20	856.0	762.0 To 915.0
Leukemia	10	2	28	45.0	41.0 To 48.0

Table 3: Average Number Of Features Having Non Zero Sore Using Modified L1-SVM

4 Method Comparison

4.1 Accuracy As Function Of Number Of Features

In this section, the accuracy of a Linear L2 SVM as a function of the number of selected features on the leukemia and Arcene datasets for the various feature selection methods (The Golub score, Modified L1- SVM selection and RFE-L2SVM) is derived. Estimation of classifier performance is un-biased by using pipelining functionality of scikit learn. Implementation regarding experiment is shown in appendix D.1

Number of selected features are taken on logarithmic scale and accuracy can be calculated for all 3 feature selection methods on both datasets. Tables - 4 & 5 - shows made from experiment. Graphs - 2 & 3 - shows pattern of accuracy for all selection methods.

Number Of Features	Accuracy Using Golub Score	Accuracy Using RFE-SVM	Accuracy Using Modified L1 SVM
1	0.68984	0.56003	0.62944
2	0.68484	0.51612	0.65381
4	0.67885	0.48510	0.66383
8	0.66423	0.46535	0.68034
16	0.65459	0.47010	0.69959
32	0.66434	0.50112	0.76400
64	0.67447	0.60055	0.77062
128	0.69509	0.64956	0.71472
256	0.68997	0.76563	0.74863
512	0.77950	0.84529	0.83966
1024	0.79938	0.87993	0.81402
2048	0.84942	0.91445	0.84954
4096	0.84454	0.90419	0.86993
8192	0.88418	0.89407	0.88406

Table 4: Accuracy As Function Of Number Of Feature : Arcene Dataset

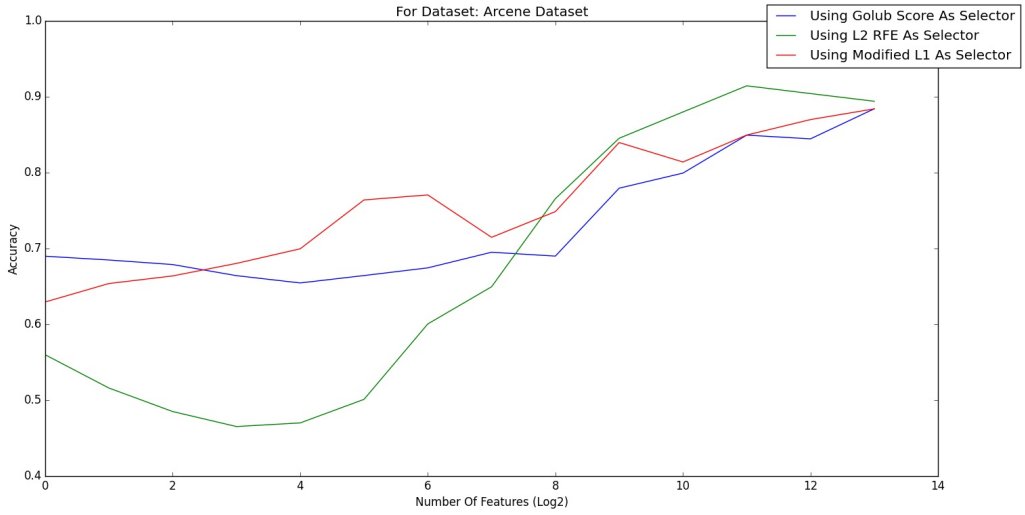

Figure 2: Accuracy Representation As Function Of Selected Features : Arcene Dataset

According to functionality of L2-SVM, it accounts square of number of slack variables. Score of each feature in L2-SVM will also be inclined to that fact. So it may be possible that for small number of features, accuracy may not be much higher.

Such kind of pattern is observed in above shown graph. Above graph suggests that after some threshold, accuracy increases exponentially & becomes higher than other selection methods if number of feature selected becomes higher than threshold. Accuracy increases in other selection methods gradually.

Number Of Features	Accuracy Using Golub Score		Accuracy Using RFE-SVM		Accuracy Using Modified L1 SVM
1	0.93047		0.80761		0.70952
2	0.93047		0.86285		0.76190
4	0.94476		0.86190		0.80571
8	0.95904		0.87619		0.94571
16	0.91619		0.94571		0.95809
32	0.97238		0.94571		0.97238
64	0.91714		0.90380		0.89047
128	0.94476		0.91714		0.91809
256	0.94476		0.88952		0.88952
512	0.95904		0.87523		0.83523
1024	0.91714		0.84761		0.87619
2048	0.89047		0.76476		0.87523
4096	0.81904		0.80571		0.86993

Table 5: Accuracy As Function Of Number Of Feature : Leukemia Dataset

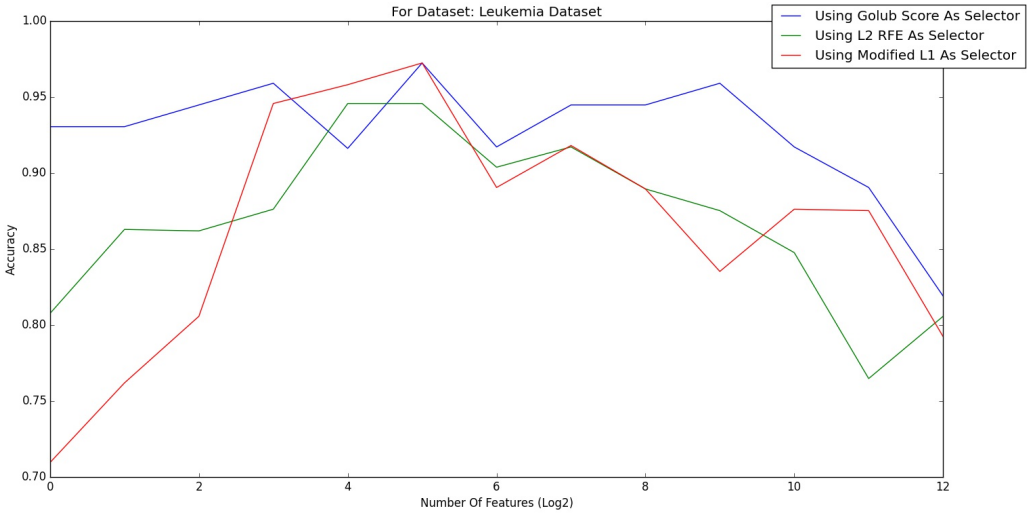

Figure 3: Accuracy Representation As Function Of Selected Features : Leukemia Dataset

Initial section, in this report, suggests that, only 40-60 features are actually relevant for Leukemia dataset. Similar behaviour is observed here. Increasing features after some threshold, for any classifier, decreases accuracy.

4.2 Model Selection

For above experiment, internal parameters of classifiers were not changed. Internal cross-validation for selecting the soft margin constant C should be performed to find optimal solution of parameters for selector and classifier to achieve higher accuracy.

According to previous observations, for Arcene dataset, high number of features and for Leukemia dataset low number of features good accuracy. So for calculating cross validation to select soft margin parameter, high number of features will be taken into account for Arcene dataset and low number of features will be taken for Leukemia dataset.

Implementation regarding this experiment is shown in appendix - D.2. Below shown both tables - 6 & 7 - depicts observation made during analysis. Bothe table suggests that, parameters are tuned finely and giving good accuracy compare to default ones.

Number Of Features	**Golub Score**		**Modified L1 SVM**		**RFE-SVM**		**Best Parameter**
	Best C Value	**Accuracy**	**Best C Value**	**Accuracy**	**Best C Value**	**Accuracy**	
512	0.01	0.78	0.0001	0.8	0.0001	0.85	**RFE With 2048 Number Of Features With C = 0.0001**
1024	0.001	0.8	0.0001	0.825	0.0001	0.885	
2048	0.001	0.855	0.0001	0.87	**0.0001**	**0.92**	
4096	0.01	0.85	0.0001	0.885	0.0001	0.91	
8192	0.0001	0.885	0.0001	0.89	0.0001	0.895	

Table 6: Best Model - Arcene Dataset

According to above table, Best combination to achieve higher accuracy in Arcene dataset is to use L2-SVM as classifier, and RFE as selector with selecting 2048 number of features and C equals to 0.0001.

Number Of Features	**Golub Score**		**Modified L1 SVM**		**RFE-SVM**		**Best Parameter**
	Best C Value	**Accuracy**	**Best C Value**	**Accuracy**	**Best C Value**	**Accuracy**	
1	1	0.93056	0.0001	0.75	0.0001	0.80555	**L1 Modified With 16 Number Of Features With C = 0.0001**
2	0.0001	0.93056	0.0001	0.81944	0.0001	0.875	
4	0.1	0.94444	0.0001	0.90277	0.0001	0.88889	
8	0.0001	0.95833	0.0001	0.93055	0.0001	0.875	
16	0.0001	0.95833	**0.0001**	**0.97222**	0.0001	0.94444	
32	0.0001	0.9722	0.0001	0.944	0.0001	0.944	

Table 7: Best Model - Leukemia Dataset

Likewise for Leukemia dataset, if L2 classifier is needed to be used, then selector of future should be modified L1-SVM with C = 0.0001. Selector should select 16 features only to achieve higher accuracy. Probable reasons for such low number is, many feature may have been inter-dependant.

Appendices

A Implementation To Parse Dataset

```
def generate_data(flag):
   if (flag == 1):
      print "For Arcene Dataset"

      y_trainData=np.genfromtxt("https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/
                                ARCENE/arcene_train.labels")
      X_trainData=np.genfromtxt("https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/
                                ARCENE/arcene_train.data")

      y_validData=np.genfromtxt("https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/
                                arcene_valid.labels")
      X_validData=np.genfromtxt("https://archive.ics.uci.edu/ml/machine-learning-databases/arcene/
                                ARCENE/arcene_valid.data")

      X = np.concatenate((X_trainData, X_validData), axis=0)
      y = np.concatenate((y_trainData, y_validData), axis=0)

      print "After Concate: X Size: ", X.shape
      print "After Concate: y Size: ", y.shape

   else:
      print "For Leukemia Dataset"

      data_train = svmlight_format.load_svmlight_file('leu.bz2')
      data_test = svmlight_format.load_svmlight_file('leu.t.bz2')

      X_train = data_train[0].toarray()
      y_train = data_train[1]

      X_test = data_test[0].toarray()
      y_test = data_test[1]

      X = np.concatenate((X_train, X_test), axis=0)
      y = np.concatenate((y_train, y_test), axis=0)

      print "After Concate: X Size: ", X.shape
      print "After Concate: y Size: ", y.shape

   return X, y
if __name__=='__main__' :
   X_Arcene,y_Arcene = generate_data(1)
   X_L, y_L = generate_data(2)
```

B Filter Method Implementation

B.1 Implementation Of Golub Score

```
def golub_score(X, y):
   flag = 1
   if(flag ==1):
      X_Positive = []
      X_Negative = []
      y_Positive = []
      y_Negative = []

      for i in range(len(y)):
         if(y[i] == -1):
            y_Negative.append(y[i])
            X_Negative.append(X[i])
         elif(y[i] == 1):
            y_Positive.append(y[i])
            X_Positive.append(X[i])

      X_Positive = np.asarray(X_Positive)
      X_Negative = np.asarray(X_Negative)
      y_Positive = np.asarray(y_Positive)
      y_Negative = np.asarray(y_Negative)

      print "Positive Datasize",X_Positive.shape
      print "Negative Datasize",X_Negative.shape
      print "Positive Labels",y_Positive.shape
      print "Negative Labels",y_Negative.shape

      Average_Positive_Example = np.average(X_Positive, axis = 0)
      Average_Negative_Example = np.average(X_Negative, axis = 0)

      Standard_Positive_Example = np.std(X_Positive, axis=0)
      Standard_Negative_Example = np.std(X_Negative, axis=0)

      print "Average Positive", Average_Positive_Example.shape
      print "Average Negative", Average_Negative_Example.shape
      print "Standard Positive", Standard_Positive_Example.shape
      print "Standard Negative", Standard_Negative_Example.shape

      Average_Difference = np.absolute(Average_Positive_Example - Average_Negative_Example)
      Standard_Sum_Temporary = Standard_Positive_Example + Standard_Negative_Example

      Standard_Sum = []
      for x in Standard_Sum_Temporary:
         if(x == 0):
            x = 1
         Standard_Sum.append(x)

      Golub = Average_Difference / Standard_Sum
      return Golub, Golub
if __name__=='__main__' :
   X_Arcene,y_Arcene = generate_data(1)
   golub_rank_arcene = golub_score(X_Arcene, y_Arcene)
   print golub_rank_arcene
   X_L, y_L = generate_data(2)
   golub_rank_L = golub_score(X_L, y_L)
```

```
print golub_rank_L
```

C Embedded Methods Implementation

C.1 Implementation To Compute Non-Zero Weight Vector Coefficients

```
def L1_SVM(X, y, flag):
   print "**Calculating Number Of Non Zero Weight Features**"
   if flag == 1:
      print "For Arcene Dataset"
   else:
      print "For Leukemia Dataset"
   run_svm = 10
   none_zero_counters = []
   for count_down in range((run_svm)):
      print "Iteration: ", count_down
      classifier = LinearSVC(penalty='l1', dual=False)

      classifier = classifier.fit(X, y)

      counter_of_needed_feature = 0

      for coef in classifier.coef_[0]:
         if(coef !=0):
            counter_of_needed_feature = counter_of_needed_feature + 1

      print "Total Non Zero Weight Features", counter_of_needed_feature
      none_zero_counters.append(counter_of_needed_feature)
   print "Average Number Of Non Zero ", round(np.average(none_zero_counters))
   print "Range", round(np.min(none_zero_counters)), " To ", round(np.max(none_zero_counters))

if __name__=='__main__' :
   X_Arcene,y_Arcene = generate_data(1)
   print "X_Arcene Size", X_Arcene.shape
   L1_SVM(X_Arcene, y_Arcene, 1)

   X_L, y_L = generate_data(2)
   L1_SVM(X_L, y_L, 2)
```

C.2 Implementation Of L1 SVM On All Features

```
def L1_SVM_Accuracy(X, y, flag):
   print "**Calculating Accuracy Of L1_SVM**"
   if flag == 1:
      print "For Arcene Dataset"
   else:
      print "For Leukemia Dataset"

   run_svm = 5
   accuracy = []
   for count_down in range((run_svm)):
      print "Iterations: ", count_down
      classifier = LinearSVC(penalty='l1', dual=False)

      cross_valid = cross_validation.StratifiedKFold(y, 5, shuffle=True, random_state=0)

      got_accuracy = np.mean(cross_validation.cross_val_score(classifier, X, y, cv=cross_valid,
                                          scoring='accuracy'))

      accuracy.append(got_accuracy)
      print "Accuracy", got_accuracy
   print "Average Accuracy", np.average(accuracy)
if __name__=='__main__' :
   X_Arcene,y_Arcene = generate_data(1)
   L1_SVM_Accuracy(X_Arcene, y_Arcene, 1)

   X_L, y_L = generate_data(2)
   L1_SVM_Accuracy(X_L, y_L, 2)
```

C.3 Implementation Of L2 SVM On Features Selected By L1 SVM

```
def L1_SVM_Selects_L2_Accuracy(X, y, flag):
   print "**Calculating Feature Selection Of L1_SVM**"
   print "L2 SVM On Feature Selected By L1"
   if flag == 1:
      print "For Arcene Dataset"
   else:
      print "For Leukemia Dataset"

   classifier = LinearSVC(penalty='l1', dual=False)

   cross_valid = cross_validation.StratifiedKFold(y, 5, shuffle=True, random_state=0)

   selector = RFECV(classifier, step=0.1)
   selector = selector.fit(X, y)

   classifier_L2 = LinearSVC(penalty='l2', dual=False)

   pipeline = make_pipeline(selector, classifier_L2)

   got_accuracy = np.mean(cross_validation.cross_val_score(pipeline, X, y, cv=cross_valid,
                                                           scoring='accuracy'))

   print "Accuracy", got_accuracy

if __name__=='__main__' :
   print "Starting Execution"
   print "Working On Arcene"
   X_Arcene,y_Arcene = generate_data(1)

   L1_SVM_Selects_L2_Accuracy(X_Arcene, y_Arcene, 1)

   print "Done On Arcene"
   print "Working On Leukemia"
   X_L, y_L = generate_data(2)

   L1_SVM_Selects_L2_Accuracy(X_L, y_L, 2)

   print " Whole Program Done!..."
```

C.4 Implementation Of L2 SVM On All Features

```
def L2_SVM_Accuracy_All(X, y, flag):

   print "**Calculating Accuracy Of L2_SVM**"
   print "L2 SVM On All Features"
   if flag == 1:
      print "For Arcene Dataset"
   else:
      print "For Leukemia Dataset"

   run_svm = 1
   accuracy = []
   for count_down in range((run_svm)):
      print "Iterations: ", count_down
      classifier = LinearSVC(penalty='l2', dual=False)

      classifier = classifier.fit(X, y)

      cross_valid = cross_validation.StratifiedKFold(y, 5, shuffle=True, random_state=0)

      got_accuracy = np.mean(cross_validation.cross_val_score(classifier, X, y, cv=cross_valid,
                                                scoring='accuracy'))

      accuracy.append(got_accuracy)
      print "Accuracy", got_accuracy
   print "Average Accuracy", np.average(accuracy)

if __name__=='__main__' :
   print "Starting Execution"
   print "Working On Arcene"
   X_Arcene,y_Arcene = generate_data(1)

   L2_SVM_Accuracy_All(X_Arcene, y_Arcene, 1)

   print "Done On Arcene"
   print "Working On Leukemia"
   X_L, y_L = generate_data(2)

   L2_SVM_Accuracy_All(X_L, y_L, 2)

   print " Whole Program Done!..."
```

C.5 Implementation Of L2 SVM On Features Selected By RFECV

```
def L2_SVM_Selects_L2_Accuracy(X, y, flag):

   print "**Calculating Accuracy Of L2_SVM By Selecting Feature By L2**"
   print "L2 SVM On Feature Selected By L2"
   if flag == 1:
      print "For Arcene Dataset"
   else:
      print "For Leukemia Dataset"

   run_svm = 1
   accuracy = []
   for count_down in range((run_svm)):
      print "Iterations: ", count_down
      classifier = LinearSVC(penalty='l2', dual=False)

      cross_valid = cross_validation.StratifiedKFold(y, 5, shuffle=True, random_state=0)

      selector = RFECV(classifier, step=0.1)
      selector = selector.fit(X, y)

      pipeline = make_pipeline(selector, classifier)

      got_accuracy = np.mean(cross_validation.cross_val_score(pipeline, X, y, cv=cross_valid,
                                                  scoring='accuracy'))

      accuracy.append(got_accuracy)
      print "Accuracy", got_accuracy

   print "Average Accuracy", np.average(accuracy)

if __name__=='__main__' :
   print "Starting Execution"
   print "Working On Arcene"
   X_Arcene,y_Arcene = generate_data(1)

   L2_SVM_Selects_L2_Accuracy(X_Arcene, y_Arcene, 1)

   print "Done On Arcene"
   print "Working On Leukemia"
   X_L, y_L = generate_data(2)

   L2_SVM_Selects_L2_Accuracy(X_L, y_L, 2)

   print " Whole Program Done!..."
```

C.6 Implementation Of Modified L1 SVM

```
def L1_SVM(X, y, flag):
   print "**Calculating Number Of None Zero Weight Features**"
   if flag == 1:
      print "For Arcene Dataset"
   else:
      print "For Leukemia Dataset"
   run_svm = 1

   score_vector_fold = np.zeros(X.shape[1])

   for count_down in range((run_svm)):
      print "Iteration: ", count_down
      classifier = LinearSVC(penalty='l1', dual=False)

      classifier = classifier.fit(X, y)

      for i in range(len(classifier.coef_[0])):
         if(classifier.coef_[0][i] !=0):
            score_vector_fold[i] = 1

   return score_vector_fold

def generateKSubSample(X, y, flag):
   k = 0
   if flag == 1:
      k = 8
   else:
      k = 2
   print "K Will Be", k
   number_rows = X.shape[0]
   number_Train = int(round(number_rows*0.8))

   rows = np.arange(number_rows)
   np.random.shuffle(rows)

   training = rows[:number_Train]
   testing = rows[number_Train:]
   y = y.reshape((-1,1))

   X_training = X[training,:]
   y_training = y[training,:]
   X_testing = X[testing,:]
   y_testing = y[testing,:]

   y_training = y_training.reshape(1, -1)[0]

   X_split_data = np.split(X_training, k)
   y_split_data = np.split(y_training, k)

   return X_split_data, y_split_data
```

```
if __name__=='__main__' :
   X_Arcene,y_Arcene = generate_data(1)

   randomization = 10
   final_counter_arcene = []
   for i in range(randomization):
      print "For Randomization", i
      X_Arcene_Split, y_Arcene_Split = generateKSubSample(X_Arcene, y_Arcene, 1)

      score_vector_arcene = np.zeros(X_Arcene.shape[1])
      for i in range(len(X_Arcene_Split)):
         print "Fold: ", i
         score_vector_arcene = score_vector_arcene + L1_SVM(X_Arcene_Split[i], y_Arcene_Split[i], 1)
      counter = 0
      for rank in score_vector_arcene:
         if rank !=0:
            counter += 1

      final_counter_arcene.append(counter)
   print "Average Non Zero Weights For Arcene Dataset", round(np.average(final_counter_arcene))

   X_L, y_L = generate_data(2)

   final_counter_L = []
   for i in range(randomization):
      print "For Randomization", i
      X_L_Split, y_L_Split = generateKSubSample(X_L, y_L, 2)

      score_vector_L = np.zeros(X_L.shape[1])
      for i in range(len(X_L_Split)):
         print "Fold: ", i
         score_vector_L = score_vector_L + L1_SVM(X_L_Split[i], y_L_Split[i], 2)
      counter = 0
      for rank in score_vector_L:
         if rank !=0:
            counter += 1

      final_counter_L.append(counter)
   print "Average Non Zero Weights For Leukemia Dataset", round(np.average(final_counter_L))
```

D Method Comparison

D.1 Accuracy Comparison

```
def rfe_feature_select(X, y, dataset):
   k = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
   if(dataset == 1):
      print "Arcene Dataset RFE Selection"
      k.append(8192)
   else:
      print "Leukemia Dataset RFE Selection"
   cross_valid = cross_validation.StratifiedKFold(y, 5, shuffle=True, random_state=0)
   accuracys = []
   for k_value in k:
      classifier = LinearSVC(penalty="l2", dual = False)
      selector = RFE(classifier, step=0.1, n_features_to_select=k_value)
      pipeline = make_pipeline(selector, classifier)
      got_accuracy = np.mean(cross_validation.cross_val_score(pipeline, X, y, cv=cross_valid,
            scoring='accuracy'))
      print "Accuracy", got_accuracy
      accuracys.append(got_accuracy)
   return k, accuracys
def L1_Modified_Features(X, y):
   X_Split, y_Split = generateKSubSample(X, y)

   score_vector = np.zeros(X.shape[1])

   for i in range(len(X_Split)):
      score_vector = score_vector + L1_SVM_Coef(X_Split[i], y_Split[i])
   return score_vector, score_vector
def modified_L1_feature_select(X, y, dataset):
   k = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
   if(dataset == 1):
      print "Arcene Dataset Modified L1 Selection"
      k.append(8192)
   else:
      print "Leukemia Dataset Modified L1 Selection"
   cross_valid = cross_validation.StratifiedKFold(y, 5, shuffle=True, random_state=0)

   results = []
   for k_value in k:
      select = SelectKBest(L1_Modified_Features, k_value)
      classifier = LinearSVC(penalty="l2", dual = False)
      pipeline = make_pipeline(select, classifier)
      gotAccuracy = np.mean(cross_validation.cross_val_score(pipeline, X, y, cv=cross_valid,
                  scoring='accuracy'))
      print "Accuracy Using L1 Modified", gotAccuracy
      results.append(gotAccuracy)
   return k, results
def compute_accuracy_of_selector(custom_selector,selector_name,classifier_name,number_feature,X,y):
   k = number_feature

   classifier = LinearSVC(penalty=classifier_name, dual=False)
   cross_valid = cross_validation.StratifiedKFold(y, 5, shuffle=True, random_state=0)

   if selector_name == "Golub":
      select = SelectKBest(custom_selector, k)
      pipeline = make_pipeline(select, classifier)
```

```
      gotAccuracy = np.mean(cross_validation.cross_val_score(pipeline, X, y, cv=cross_valid,
                            scoring='accuracy'))
      return gotAccuracy
def golub_feature_select(X, y, dataset):
   k = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
   if(dataset == 1):
      print "Arcene Dataset Golub Selection"
      k.append(8192)
   else:
      print "Leukemia Dataset Golub Selection"
   results = []
   for k_value in k:
      accuracy_by_golub = compute_accuracy_of_selector(golub_score, "Golub", "l2", k_value, X, y)
      results.append(accuracy_by_golub)
   return k, results
if __name__=='__main__' :
   X_Arcene,y_Arcene = generate_data(1)

   kValuesArceneGolub, accuracyValuesOfGolubArcene = golub_feature_select(X_Arcene, y_Arcene, 1)
   kValuesArceneRFE, accuracyValuesOfRFEArcene = rfe_feature_select(X_Arcene, y_Arcene, 1)
   kValuesArceneModified, accuracyValuesOfModifiedArcene =
   modified_L1_feature_select(X_Arcene, y_Arcene, 1)
   logOfK = np.log2(kValuesArceneRFE)

   plot_data(logOfK, accuracyValuesOfGolubArcene, accuracyValuesOfRFEArcene,
   accuracyValuesOfModifiedArcene, "Arcene Dataset")

   X_L, y_L = generate_data(2)
   kValuesLGolub, accuracyValuesOfGolubL =golub_feature_select(X_L, y_L, 2)
   kValuesLRFE, accuracyValuesOfRFEL = rfe_feature_select(X_L, y_L, 2)
   kValuesLModified, accuracyValuesOfModifiedL = modified_L1_feature_select(X_L, y_L, 2)
   logOfKL = np.log2(kValuesLGolub)
   plot_data(logOfKL, accuracyValuesOfGolubL, accuracyValuesOfRFEL, accuracyValuesOfModifiedL,
   "Leukemia Dataset")
```

D.2 Model Selection

```
def model_selection(X, y, flag):
   k = []
   if flag == 1:
      k = [512, 1024, 2048, 4096, 8192]
   else:
      k = [1, 2, 4, 8, 16]
   classifier = LinearSVC(penalty = 'l2', dual=False)
   cross_valid = cross_validation.StratifiedKFold(y, 5, shuffle=True, random_state=0)
   best_global_accuracy = 0
   best_golub_accuracy = 0
   best_golub_answer = ""
   best_global_answer = ""
   Cs = np.logspace(-4, 3, 8)
   dict_Cs = {'SVMlinear2__C':Cs}

   k_golub= k

   for features in k_golub:
      print "Running Golub For Features", features
      selector_golub = SelectKBest(golub_score, features)

      pipeline_information = [('golub',selector_golub),('SVMlinear2',classifier)]
      pipeline_golub = Pipeline(pipeline_information)

      grid_search_golub = GridSearchCV(estimator = pipeline_golub, param_grid=dict_Cs, cv=cross_valid)
      grid_search_golub.fit(X, y)

      got_accuracy = grid_search_golub.best_score_
      print got_accuracy
      print grid_search_golub.best_params_

      if got_accuracy>=best_golub_accuracy:
         best_golub_answer = "Feature " + str(features) + " Parameter" + str(grid_search_golub.best_params_)
         best_golub_accuracy = got_accuracy
      if got_accuracy>=best_global_accuracy:
         best_global_answer = "Golub Feature " + str(features) + " Parameter" + str(grid_search_golub.best_pa
         best_global_accuracy = got_accuracy
   print "Best Combination Global: ", best_global_answer
   print "Best Global Results: ", best_golub_answer
   best_RFE_accuracy = 0
   best_RFE_answer = ""

   k_rfe = k
   for features in k_rfe:
      print "Running RFE For Features", features

      selector_RFE = RFE(classifier, step=0.1, n_features_to_select=features)

      pipeline_information = [('RFE',selector_RFE),('SVMlinear2',classifier)]
      pipeline_RFE = Pipeline(pipeline_information)

      grid_search_RFE = GridSearchCV(estimator = pipeline_RFE, param_grid=dict_Cs, cv=cross_valid)
      grid_search_RFE.fit(X, y)

      got_accuracy = grid_search_RFE.best_score_

      print got_accuracy
```

```
      print grid_search_golub.best_params_

      if got_accuracy>=best_RFE_accuracy:
         best_RFE_answer = "Feature " + str(features) + " Parameter" + str(grid_search_golub.best_params_)
         best_RFE_accuracy = got_accuracy
      if got_accuracy>=best_global_accuracy:
         best_global_answer = "RFE Feature " + str(features) + " Parameter" + str(grid_search_golub.best_para
         best_global_accuracy = got_accuracy
   print "Best Combination Global: ", best_global_answer
   print "Best RFE Results: ", best_RFE_answer
   best_L1M_accuracy = 0
   best_L1M_answer = ""

   k_l1m = k
   for features in k_l1m:
      print "Running L1 Modified For Features", features

      selector_LM = SelectKBest(L1_Modified_Features, features)

      pipeline_information = [('modified',selector_LM),('SVMlinear2',classifier)]
      pipeline_LM = Pipeline(pipeline_information)

      grid_search_LM = GridSearchCV(estimator = pipeline_LM, param_grid=dict_Cs, cv=cross_valid)
      grid_search_LM.fit(X, y)

      got_accuracy = grid_search_LM.best_score_

      print got_accuracy
      print grid_search_golub.best_params_

      if got_accuracy>=best_L1M_accuracy:
         best_L1M_answer = "Feature " + str(features) + " Parameter" + str(grid_search_golub.best_params_)
         best_L1M_accuracy = got_accuracy
      if got_accuracy>=best_global_accuracy:
         best_global_answer = "L1 Modified Feature " + str(features) + " Parameter" + str(grid_search_golub.b
         best_global_accuracy = got_accuracy
   print "Best Combination Global: ", best_global_answer
   print "Best L1 Modified Results: ", best_L1M_answer

if __name__=='__main__' :
   X_Arcene,y_Arcene = generate_data(1)
   model_selection(X_Arcene, y_Arcene, 1)

   X_L, y_L = generate_data(2)
   model_selection(X_L, y_L, 2)
```