Project: Sequence-based fusion method for identify disease genes.

The identification of disease genes from human genome is of great importance to improve diagnosis and treatment of disease. Identifying disease associated genes from the vast number of candidates using experimental methods is an expensive and time consuming task. Hence, the need of computational approaches has been emerged.

Many machine learning (ML) algorithms have been applied to identify disease genes. Most of these methods regarded this problem as binary-class classification problem.

The main chal-lenges are summarized in:

- Selecting the more complete prior-knowledge about genes to generate the feature vectors.
- Selecting negative data from unknown genes to build and evaluate the classification model.
- Selecting the proper classi-fication method.

In this project, we use a sequence-based fusion method to identify disease genes using solely the primer structure of the proteins as a prior-knowledge. Four different feature representation methods are used to transform the amino acids sequences to numerical feature vectors. Then, six sequence-based individual classifiers using SVM algorithm have been employed. The outputs of these SVM-based predictors were fused using C4.5 algorithm. In addition, a significant improvement in the classification performance has been also observed by using fusion of SVM-based predictors.

We first download standard packages.

In [13]:
```
import numpy as np
from sklearn import svm
import pandas as pd
import csv
import time
from datetime import datetime
from sklearn import tree
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import os
```

Load the Data

Unknown genes with similar features to the confirmed disease genes could be a candidate disease gene with high probability. The similarity between unknown and known disease genes is based on a variant genomic data which has been generated using high-throughput technologies. Here, the motivation is that the proteins which lead to similar phenotypes have a higher chance of being connected in the network, in this way, we can prioritize candidate disease genes and render a fewer prioritized list for further investigations using functional similarity data.

We load the functional similarity data.

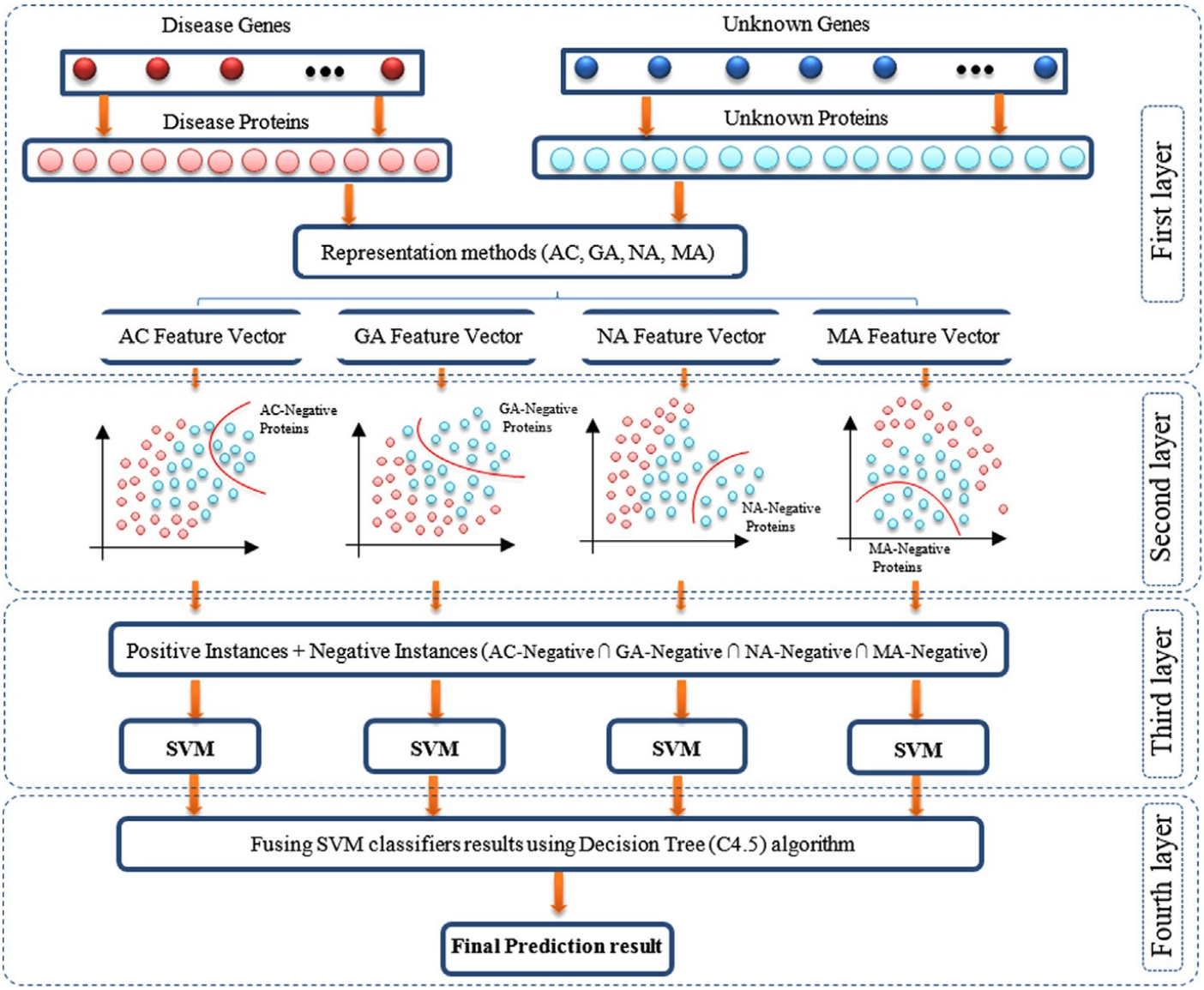
In [14]:
```
ConnectDate = np.loadtxt('knowndiseasemirnainteraction.txt',dtype=int)-1
# load the similar matrix of the disease
DS1 =np.loadtxt('SimilarMatrix1.txt')
DS2 = np.loadtxt('SimilarMatrix2.txt')
FS = np.loadtxt(r'miRNASimilarMatrix.txt')
```

There are three key variables:

- `nm` = number of miRNA.
- `nd` = number of disease.
- `nc` = number of miRNA-disease.

In [15]:
```
nm = 495 #number of miRNA.
nd = 383 #number of disease.
nc = 5430 #number of miRNA-disease
```

Sequence-based fusion method



As the figure shows, our method for dentifying and prioritizing disease genes consists of four steps:

- Translate corresponding gene products (proteins) into four numerical feature vectors using four types of protein sequence translator;
- Selecting negative data from unknown genes;
- Modeling each feature vector using SVM algorithm;
- Decision Tree algorithm is used to make the final decision by fusing the predicting results of the base SVM classifiers. To evaluate the performance of item recommendation, we adopted the leave-one-out evaluation, which has been widely used.

In [16]:
```
A=np.zeros((nd,nm),dtype=float)  #create matrix 383*495
for i in range(nc):
    A[ConnectDate[i,1], ConnectDate[i,0]] = 1
globalrank_pos = []
localrank_pos = []
predict_0_local = []
accList = []
```

Protein sequence translator

One of the most important challenges in identifying disease gene problem using machine learning algorithm is to extract feature vectors for disease and unknown genes. In this project, we use corresponding gene products (Proteins) to characterize genes.

In [17]:
```
i_nc = 0
def Getgauss_miRNA(adjacentmatrix,nm):
    KM = np.zeros((nm,nm))
    gamaa=1
    sumnormm=0
    for i in range(nm):
        normm = np.linalg.norm(adjacentmatrix[:,i])**2
        sumnormm = sumnormm + normm
    gamam = gamaa/(sumnormm/nm)
    for i in range(nm):
          for j in range(nm):
                  KM[i,j]= np.exp (-gamam*(np.linalg.norm(adjacentmatrix[:,i]-
adjacentmatrix[:,j])**2))
    return KM

# Getgauss_disease similar matrix
def Getgauss_disease(adjacentmatrix,nd):
    KD = np.zeros((nd,nd))
    gamaa=1
    sumnormd=0
    for i in range(nd):
        normd = np.linalg.norm(adjacentmatrix[i])**2
        sumnormd = sumnormd + normd
    gamad=gamaa/(sumnormd/nd)
    for i in range(nd):
        for j in range(nd):
            KD[i,j]= np.exp(-(gamad*(np.linalg.norm(adjacentmatrix[i]-adjacent
matrix[j])**2)))
    return KD

T3 = time.time()
A[ConnectDate[i_nc,1],ConnectDate[i_nc,0]] = 0  # leave-one-out
KM = Getgauss_miRNA(A,nm)  #Recalculating Gauss Similarity Matrix of miRNA wit
h Adjacency Matrix
KD = Getgauss_disease(A,nd)  #Recalculating Gauss Similarity Matrix of disease
 with Adjacency Matrix
positive_sample_index = np.argwhere(A == 1)#positive sample
unknown_sample_index =  np.argwhere(A == 0)# negative sample
for i in range(unknown_sample_index.shape[0]):
    if unknown_sample_index[i,0]== ConnectDate[i_nc,1] and unknown_sample_inde
x[i,1]== ConnectDate[i_nc,0]:
        i_1_0 = i
        break
```

Selecting negative data

After generating the feature vectors for all genes using repre-sentation methods, it is required to select a negative protein set from the unknown proteins to build a dataset with both positive and reliable negative instances.

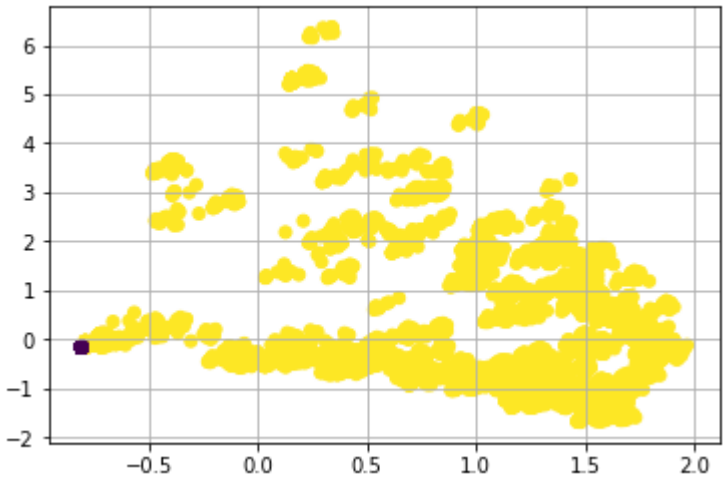
In [18]:
```
def sample_partition(positive_sample_index,unknown_sample_index,FeatureD,Featu
reM):
    positive_sample_FetureD = FeatureD[positive_sample_index[:,0]]
    positive_sample_FetureM = FeatureM[positive_sample_index[:,1]]
    positive_sample_Feture = np.hstack((positive_sample_FetureD,positive_sampl
e_FetureM))
    unknown_sample_FeatureD = FeatureD[unknown_sample_index[:,0]]
    unknown_sample_FeatureM = FeatureM[unknown_sample_index[:,1]]
    unknown_sample_Feature = np.hstack((unknown_sample_FeatureD,unknown_sample
_FeatureM))
    mean=np.mean(positive_sample_Feture,0)

    T1=time.time()
    distance=[]
    for i in range(unknown_sample_Feature.shape[0]):
        dis=np.dot(unknown_sample_Feature[i],mean)/ np.linalg.norm(unknown_sam
ple_Feature[i])/np.linalg.norm(mean)
        distance.append(dis)
    T2 = time.time()
    distance = np.array(distance)
    arg_distance = np.argsort(distance)
    negitive_sample_index = arg_distance[0:nc-1]
    negitive_sample_feature = unknown_sample_Feature[negitive_sample_index]
    test_sample_index = unknown_sample_index
    test_sample_feature = unknown_sample_Feature
    return positive_sample_Feture,negitive_sample_feature,test_sample_feature

#SVM1
T4 = time.time()
positive_sample_DS1_FS,negative_sample_DS1_FS,test_sample_DS1_FS = sample_part
ition(positive_sample_index,unknown_sample_index,DS1,FS)
train_sample_DS1_FS = np.vstack((positive_sample_DS1_FS,negative_sample_DS1_FS
))
X_train = np.zeros((len(train_sample_DS1_FS),6))
X_test = np.zeros((len(test_sample_DS1_FS),6))
Y_value = np.zeros((len(train_sample_DS1_FS),6))
```

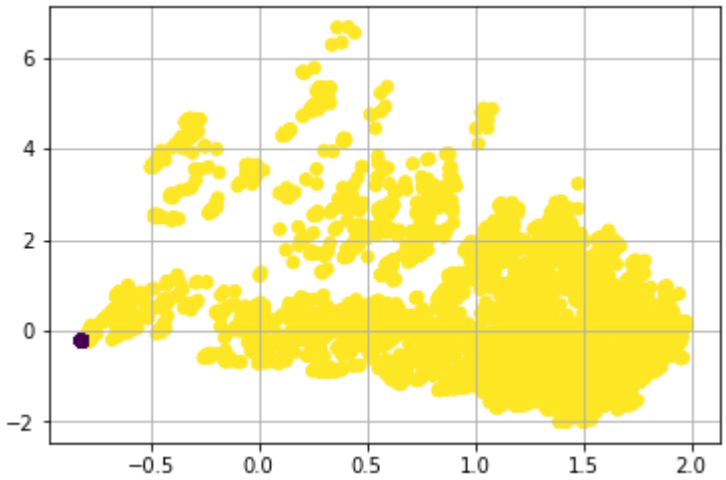
SVM algorithm

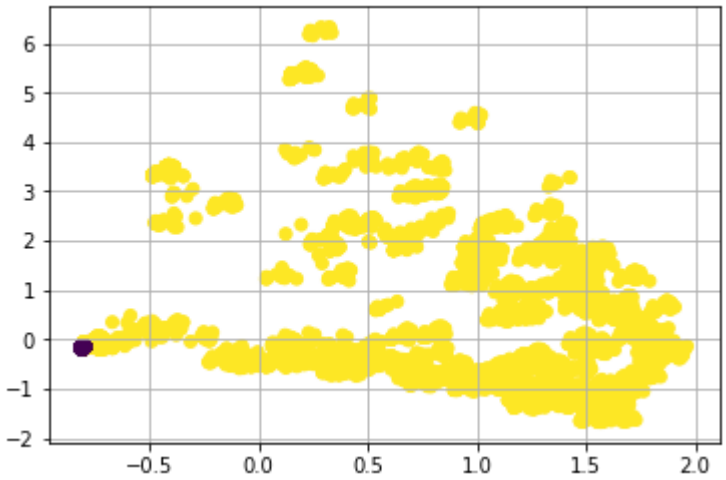
Support Vector Machine (SVM) is a popular and promising method for data classification in many application areas. And we visualize data using PCA.

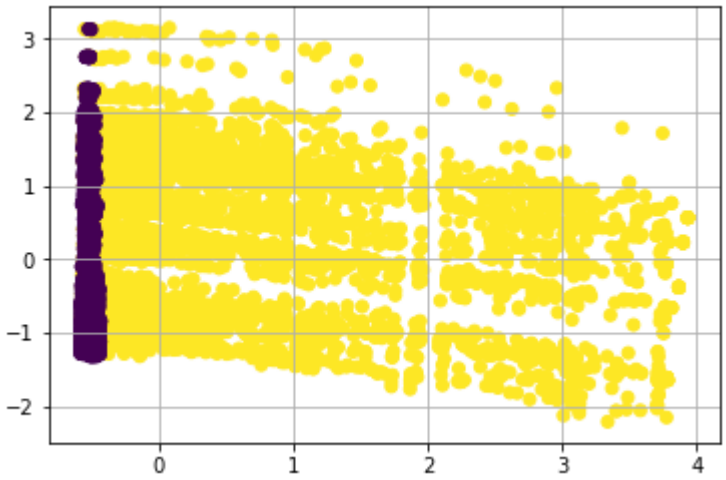
In [19]:
```
def train_and_predict(train_sample_feature,test_sample_feature,X_train,X_test,
Y_value,svmID,accList):
    ncomp = 6
    pca =PCA(n_components=ncomp, svd_solver='randomized',whiten=True ).fit(tra
in_sample_feature)
    Z1 = pca.transform(train_sample_feature)
    pca =PCA(n_components=ncomp, svd_solver='randomized',whiten=True ).fit(tes
t_sample_feature)
    Z2 = pca.transform(test_sample_feature)
    clf=svm.SVC()
    train_sample_lable = [1 for j in range(5430)]+[0 for j in range(train_samp
le_feature.shape[0]-5430)]
    clf.fit(Z1,train_sample_lable)
    test_result =clf.decision_function(Z2)
    X_train[:,svmID] = clf.decision_function(Z1)
    #X_train[:,1] = train_result[:,4]
    Y_value = train_sample_lable
    X_test[:,svmID] = test_result

    ncomp = 2
    pca =PCA(n_components=ncomp, svd_solver='randomized',whiten=True ).fit(tra
in_sample_feature)
    Z = pca.transform(train_sample_feature)
    colors = train_sample_lable
    plt.grid()
    plt.scatter(Z[:,0],Z[:,1],c = colors)
    plt.show()

    accList.append(clf.score(Z1,train_sample_lable))

    return(X_train,X_test,Y_value,test_result,accList)


(X_train,X_test,Y_value,test_result_DS1_FS,accList) = train_and_predict (train
_sample_DS1_FS,test_sample_DS1_FS,X_train,X_test,Y_value,0,accList)
T5 = time.time()

 #SVM2
positive_sample_DS2_FS,negative_sample_DS2_FS,test_sample_DS2_FS = sample_part
ition(positive_sample_index,unknown_sample_index,DS2,FS)
train_sample_DS2_FS = np.vstack((positive_sample_DS2_FS,negative_sample_DS2_FS
))
(X_train,X_test,Y_value,test_result_DS2_FS,accList) = train_and_predict (train
_sample_DS2_FS,test_sample_DS2_FS,X_train,X_test,Y_value,1,accList)

  #SVM3
positive_sample_KD_FS,negative_sample_KD_FS,test_sample_KD_FS = sample_partiti
on(positive_sample_index,unknown_sample_index,KD,FS)
train_sample_KD_FS = np.vstack((positive_sample_DS1_FS,negative_sample_KD_FS))

(X_train,X_test,Y_value,test_result_KD_FS,accList) = train_and_predict (train_
sample_KD_FS,test_sample_KD_FS,X_train,X_test,Y_value,2,accList)

  #SVM4
positive_sample_DS1_KM,negative_sample_DS1_KM,test_sample_DS1_KM = sample_part
ition(positive_sample_index,unknown_sample_index,DS1,KM)
```

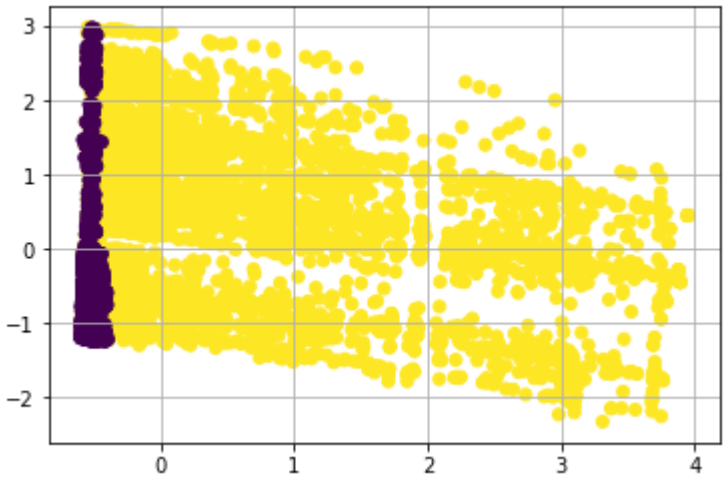
```
train_sample_DS1_KM = np.vstack((positive_sample_DS1_KM,negative_sample_DS1_KM
))
(X_train,X_test,Y_value,test_result_DS1_KM,accList) = train_and_predict (train
_sample_DS1_KM,test_sample_DS1_KM,X_train,X_test,Y_value,3,accList)

  #SVM5
positive_sample_DS2_KM,negative_sample_DS2_KM,test_sample_DS2_KM = sample_part
ition(positive_sample_index,unknown_sample_index,DS2,KM)
train_sample_DS2_KM = np.vstack((positive_sample_DS2_KM,negative_sample_DS2_KM
))
(X_train,X_test,Y_value,test_result_DS2_KM,accList) = train_and_predict (train
_sample_DS2_KM,test_sample_DS2_KM,X_train,X_test,Y_value,4,accList)

  #SVM6
positive_sample_KD_KM,negative_sample_KD_KM,test_sample_KD_KM = sample_partiti
on(positive_sample_index,unknown_sample_index,KD,KM)
train_sample_KD_KM = np.vstack((positive_sample_KD_KM,negative_sample_KD_KM))
(X_train,X_test,Y_value,test_result_KD_KM,accList) = train_and_predict (train_
sample_KD_KM,test_sample_KD_KM,X_train,X_test,Y_value,5,accList)
```

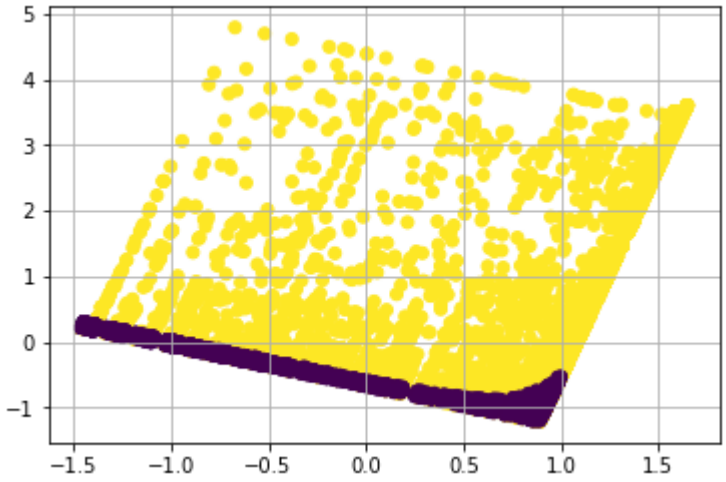

6
5
4
3
2
1
0
−1
−2
−0.5
0.0
0.5
1.0
1.5
2.0


6
4
2
0
−2
−0.5
0.0
0.5
1.0
1.5
2.0


6
5
4
3
2
1
0
−1
−2
−0.5
0.0
0.5
1.0
1.5
2.0



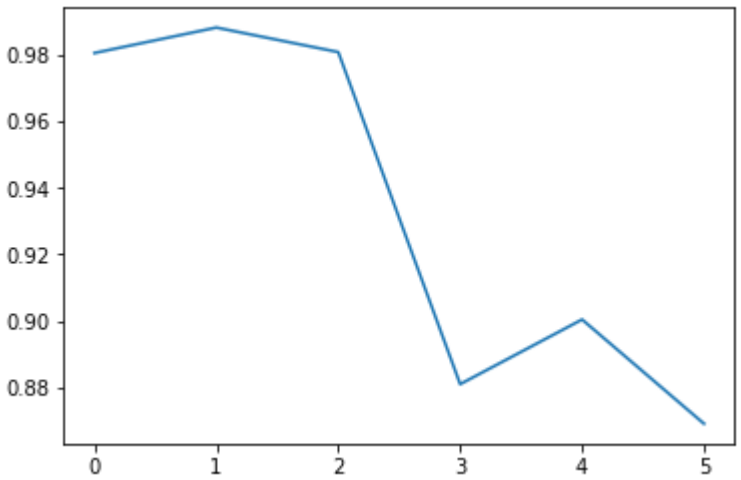




Decision Tree algorithm

Since using the same classifier (SVM) to classify the different feature vectors of the same instances, produces some uncertainties and makes some individual errors, a reasonable fusion of these classifiers are more likely reduce the overall prediction inaccura-cies and provides better prediction result. It is a powerful solution to solve tough classification problems (such as disease gene identification) which include dataset with noisy data.

In [20]:
```
clf = tree.DecisionTreeRegressor(splitter='random',min_samples_split=3,min_sam
ples_leaf = 2)
Y_value = np.zeros((len(train_sample_DS1_FS)))
for i in range(6):
    Y_value +=X_train[:,i]
Y_value = Y_value/6
clf = clf.fit(X_train, Y_value)

predict_0 = clf.predict(X_test)
predict_0_globalrank = pd.Series(predict_0).rank(ascending=False)
globalrank_pos.append(predict_0_globalrank[i_1_0])#global
j=0
for i in range(unknown_sample_index.shape[0]):
    if unknown_sample_index[i,0] == ConnectDate[i_nc,1]:
        predict_0_local.append(predict_0[i]) # render a fewer prioritized list
        j=j+1
    if i==i_1_0:
        i_local=j
predict_0_localrank=pd.Series(predict_0_local).rank(ascending=False)
localrank_pos.append(predict_0_localrank[i_local-1]) #find the local rank


A[ConnectDate[i_nc,0], ConnectDate[i_nc,1]] = 1

globalrank_posTemp = np.array(globalrank_pos)

localrank_posTemp = np.array(localrank_pos)

np.savetxt('result.dat',globalrank_posTemp)
np.savetxt('localResult.dat',localrank_posTemp)

plt.plot(accList)
plt.show()

globalrank_posTemp = np.array(globalrank_pos)
localrank_posTemp = np.array(localrank_pos)

np.savetxt('result.dat',globalrank_posTemp)
np.savetxt('localResult.dat',localrank_posTemp)
```



Conclusion

In this work, we use Sequence-based fusion method to identify disease genes. In this way, the amino acid sequence of the proteins which has been carried out to present the genes (proteins) into four different feature vectors. To select more likely negative data from candidate genes, the intersection set of four negative sets which are generated using distance approach is considered. Then, Decision Tree (C4.5) has been applied as a fusion method to combine the results of six independent state-of the-art predictors based on support vector machine (SVM) algorithm, and to make the final decision.