ECEN 649-600: Pattern Recognition Computer Project

Zhengyu Guo Wei Qiao
UIN: 521007575 522006310

Abstract:

In this project report, we carry out simulations to determine what method is good for prognosis on genetic data. Feature selection, classification rule, error estimation and number of features used are considered into this project. By presenting the error rates of different methods used for prognosis on genetic data, we give comparisons among these methods and finally we present our analysis to the results.

1. Introduction

This project aims to analyze the performance of using different classifier on a series of gene data. The classifier design includes classification rule, feature selection and error estimation. Also the number of selected features is considered. The data we use is from the paper

[1] Marc J. van de Vijver,M.J., He,Y.D., van't Veer,L.J., et al. (2002), "A gene-expression signature as a predictor of survival in breast cancer." New Eng. J. Med., 347, 1999-2009.

This paper analyzes a large number of microarrays prepared with RNA from breast tumor samples from 295 patients. Of the 295 microarrays, 115 belong to the "good-prognosis" class, whereas the remaining 180 belong to the "poor-prognosis" class. The expression data corresponds to a previously-published 70-gene signature. The gene expression data was randomly divided into a training sample set (containing 60 samples) and testing sample set (containing 235 samples). The testing samples will be used for hold-out estimation of the true classification error. The proportion of good and poor prognosis samples was kept approximately the same in the training and testing sets.

This project searches for gene feature sets that best discriminate the two prognosis classes on the training data by employing three classification rules, two error estimation criteria, and two feature selection methods. In this project, we consider three classification rules: Diagonal LDA (DLDA), 3NN, and Linear SVM (LSVM). Each feature set includes 1-5 genes which will be filtered by exhaustive search (ES) or sequential forward search (SFS) procedure based on resubstitution and Leave-one-out (LOO) cross-validation error estimators. Finally, classifiers will be generated with considering selected features based on different classification rules.

2. Method

In this project, we consider the following methods used for prognosis of genetic data. The entire procedure includes three parts: classification rule, feature selection and error estimation. Classification rule is used for designing prognosis method given a fixed model and known features. Feature selection is the process of selecting a subset of relevant features for use in classification. It is usually performed based on a given error estimator and carried out for a certain classification rule. Error estimation aims to estimate the prediction error given a certain classifier. It is involved in classifier design itself and in feature selection.

Now, we are going to search for gene feature sets that best discriminate two prognosis classes, which are represented as 1 (good prognosis) and 0 (poor prognosis), on the training data, for different classification rules, error estimation criteria, and feature selection methods.

The classification rules we will employ are:

- Diagonal LDA
- 3NN
- Linear SVM

Feature sets will be searched via wrapper feature selection, using each of the following error estimators:

- Resubstitution
- Leave-one-out cross-validation

The feature selection procedure to be employed is

- Exhaustive search (1-2 genes)
- Sequential forward search (1-5 genes)

Thus, we should determine a total of 42 methods, corresponding to the three classification rules, two error estimation criteria, two feature selection methods and different number of features used.

3. Results

In this section, we carry out simulations to compare the performances of methods depicted in section 2. The simulation programs are realized using Python. Both training error rate and testing error rate are given. Analysis of results will be given in section 4.

Table.1 The performance of the methods using exhaustive search

method	No. of features	Estimated error	Features		Test error
Exhaustive search, LOO, KNN	1	0.183333333	49		0.225531915
	2	0.1	7	66	0.212765957
Exhaustive search, Resubstitution, KNN	1	0.1	66		0.225531915
	2	0.033333333	7	66	0.212765957
Exhaustive search, LOO, SVM	1	0.133333333	49		0.187234043
	2	0.116666667	30	49	0.136170213
Exhaustive search, Resubstitution, SVM	1	0.133333333	49		0.187234043
	2	0.1	60	66	0.174468085
Exhaustive search, LOO, DLDA	1	0.25	25		0.336170213
	2	0.133333333	3	23	0.293617021
Exhaustive search, Resubstitution, DLDA	1	0.183333333	25		0.225531915
	2	0.133333333	3	23	0.212765957

Table.1 The performance of the methods using SFS

method	No. of features	Estimated error	Features					Test error
SFS,LOO,KNN	1	0.183333333	49					0.225531915
	2	0.116666667	49	37				0.195744681
	3	0.116666667	49	37	24			0.2

	4	0.083333333	49	37	24	43		0.2
	5	0.083333333	49	37	24	43	36	0.182978723
SFS, Resubstitution, KNN	1	0.1	66					0.225531915
	2	0.033333333	66	7				0.212765957
	3	0.016666667	66	7	48			0.221276596
	4	0.016666667	66	7	48	38		0.246808511
	5	0.016666667	66	7	48	38	41	0.238297872
SFS, LOO, SVM	1	0.133333333	49					0.187234043
	2	0.116666667	49	30				0.136170213
	3	0.066666667	49	30	27			0.153191489
	4	0.066666667	49	30	27	5		0.144680851
	5	0.066666667	49	30	27	5	28	0.14893617
SFS, Resubstitution, SVM	1	0.133333333	49					0.187234043
	2	0.116666667	49	27				0.195744681
	3	0.066666667	49	27	30			0.153191489
	4	0.066666667	49	27	30	21		0.136170213
	5	0.066666667	49	27	30	21	22	0.127659574
SFS, LOO, DLDA	1	0.25	25					0.29787234
	2	0.2	25	3				0.255319149
	3	0.166666667	25	3	14			0.280851064
	4	0.116666667	25	3	14	23		0.2
	5	0.1	25	3	14	23	24	0.174468085
SFS, Resubstitution, DLDA	1	0.183333333	25					0.29787234
	2	0.2	25	3				0.255319149
	3	0.166666667	25	3	42			0.25106383
	4	0.15	25	3	42	12		0.259574468
	5	0.116666667	25	3	42	12	60	0.208510638

4. Analysis and Conclusion

From the results, we could see that SVM performs the best among the three classification rules. Resubstitution is better than LOO for DLDA and KNN. But LOO is better than resubstitution for SVM.

Increasing the number of selected features does not always increase the performance of final classification performance. It dues to that the prognosis or disease expression is just related to a small number of genes (also known as features). When we find this set of features, we could obtain the optimal result. But if we continue increasing the number of selected features, the noise will be introduced into the classification. Then, the prognosis performance will decrease.

From the aspect of running speed, we could know that SVM is always the slowest method no matter in training phase or in testing phase. SVM is the most complex classification rule among the three methods. As comparison, KNN and DLDA are relatively simple. DLDA is the fastest classification method. But its performance based on the model estimated from the training data. It assumes that the data in the same class obeys the same Gaussian distribution. But when the data is not Gaussian distributed, the performance of DLDA will decrease. Meanwhile, KNN is a consistent classification rule. When we have a large set of training data, its performance will approximate the Bayes error. SVM works well when we only have a small number of training data.

For the feature selection methods given a fixed number of features, we could know that exhaustive search could find an optimal feature set for a given classification rule. But it is very slow since it should calculate all the possible combinations of features. Meanwhile, SFS is suboptimal. It can find the currently optimal feature given a formerly found feature set. We could know what this method finds is local optimal. But SFS is very fast. The complexity of exhaustive search is $O(N^m)$ where m is the number of features used, but as comparison, the complexity of SFS is $O(N)$. As the dimension of data sequence increases, the complexity of exhaustive search will become impossible to carry out. But in fact, although exhaustive search can find the optimal feature set given a certain classification rule with a lowest classification error in training step, but sometimes this training step may be over fitted. For example, for LOO-KNN methods with 2 selected features, the one using exhaustive search has a 0.1 estimated error rate which is lower than the estimated error rate 0.11667 of the one with SFS, but its testing error rate is 0.212766 which is higher than the testing error of the method using SFS which is 0.195745.

Appendix: Python Codes

Classification rules:

KNN.PY

```
from operator import itemgetter
class KNN:
        def __init__(self,K):
                self.K=K
                self.data=[]
                 self.label=[]

        def training(self,t_data,t_label):
                self.data=t_data
                 self.label=t_label

         def testing(self,vectors,labels):
                 ret=[-1]*len(labels)
                 for i in range(0,len(vectors)):
                         pre=self.testing_one(vectors[i])
                         if(pre==labels[i]):
                                 ret[i]=1
                         else:
                                 ret[i]=0
                 return ret
        def testing_one(self,test_v):
```

```
                distance=[]
                for i in range(0,len(self.data)):
                        vector=self.data[i]
                         label=self.label[i]
                        distance.append({'dist':L2_norm(vector,test_v),'label':label})
                 sort=sorted(distance,key=itemgetter('dist'))
                tmp=0.0
                for i in range(0,self.K):
                        tmp=tmp+sort[i]['label']
                tmp=tmp/self.K
                if(tmp>=0.5):
                        return 1
                else:
                        return 0

def L2_norm(X,Y):
        return sum((i-j)**2 for (i,j) in zip(X,Y))
```

SVM.py (based on SCIKIT-Learn tool box)

```
from sklearn import svm
class SVM:
    def __init__(self):
        self.SVM=svm.SVC(kernel='linear')

    def training(self,X,Y):
        self.SVM=svm.SVC(kernel='linear')
        self.SVM.fit(X,Y)

    def testing(self,vectors,labels):
        pre=self.SVM.predict(vectors)
        return [pre[i]==labels[i] for i in range(0,len(labels))]
```

DLDA.py

```
from math import log
class DLDA:
        def __init__(self):
                self.a=[]
                self.b=0
        def training(self,data,label):
                sep=self.sep_data(data,label)
                u0=self.mean(sep[0])
                u1=self.mean(sep[1])
                N0=len(sep[0])
                N1=len(sep[1])
                cov0=self.var(sep[0],u0,N0)
                cov1=self.var(sep[1],u1,N1)
                sigma_inv=self.pooled_sigma_inv(cov0,cov1,N0,N1)
                self.set_a_b(u0,u1,sigma_inv,N0,N1)

        def testing(self,data,label):
                ret=[-1]*len(label)
                for i in range(len(data)):
                        ax=0
                        for j in range(len(data[i])):
                                ax+=self.a[j]*data[i][j]
                        gn=ax+self.b
                        pre=(gn>=0)
                        if pre==label[i]:
                                ret[i]=1
                        else:
                                ret[i]=0
                return ret
```

```
    def mean(self,data):
        N=len(data[0])
        m=[0]*N
        for i in data:
            for j in range(N):
                m[j]+=i[j]
        return [float(i)/float(N) for i in m]

    def sep_data(self,data,label):
        c0=[]
        c1=[]
        for (i,j) in zip(data,label):
            if j==0:
                c0.append(i);
            elif j==1:
                c1.append(i);
        return [c0,c1]
    def var(self,data,mean,N):
        V=[0]*len(mean)
        for i in data:
            for j in range(len(mean)):
                V[j]+=(i[j]-mean[j])**2
        return [float(i)/float(N-1) for i in V]
    def pooled_sigma_inv(self,V0,V1,N0,N1):
        N=len(V0)
        return [float(N0+N1-2)/((N0-1)*V0[i]+(N1-1)*V1[i]) for i in range(N)]

    def set_a_b(self,u0,u1,sigma_inv,N0,N1):
        self.a=[sigma_inv[i]*(u1[i]-u0[i]) for i in range(len(u0))]
        bb=[sigma_inv[i]*(u1[i]+u0[i]) for i in range(len(u0))]
        b=0
        for i in range(len(u0)):
            b+=-1.0/2.0*(u1[i]-u0[i])*bb[i]
        self.b=b+log(float(N1)/N0)
```

Error estimation (including LOO and resubtitution):

```
from KNN import *
class LOO:
    def estimate(self,classifier,data_set,label):
        assert(len(data_set))
         assert(len(data_set)==len(label))
        N=len(data_set)
        testing=[data_set.pop(0)]
         t_label=[label.pop(0)]
        num=0
        for i in range(0,N):
            classifier.training(data_set,label)
            num+=sum(classifier.testing(testing,t_label))
            data_set.append(testing[0])
             label.append(t_label[0])
            testing=[data_set.pop(0)]
             t_label=[label.pop(0)]
        corr=float(num)/float(N)
        return corr
class Resub:
    def estimate(self,classifier,data_set,label):
        assert(len(data_set))
        classifier.training(data_set,label)
        N=len(data_set)
```

```
                num=sum(classifier.testing(data_set,label))
#                print  float(num)/float(N)
                return float(num)/float(N)
```

Feature selection:

```
from itertools import combinations
from init import *
from operator import itemgetter
class exhaustive:
        def search(self,classifier,estimator,training_file):
                d=read_data(training_file)
                selected_features=[]
                for i in range(1,3):
                        comb=self.exhaustive_search(70,i)
                        optimal=0
                        tmp=[]
                        for j in comb:
                                data=read_col(j,d)
                                label=read_col_label(71,d)
                                tmp_corr=estimator.estimate(classifier,data,label)
                                tmp.append({'feature':j,'corr':tmp_corr})
                                if tmp_corr > optimal:
                                        optimal=tmp_corr
                        for k in tmp:
                                if k['corr']==optimal:
                                        selected_features.append(k)
                return selected_features
        def exhaustive_search(self,total,num):
                ind=range(1,total+1)
                comb=[]
                tmp=combinations(ind,num)
                for j in tmp:
                        comb.append(list(j))
                return comb
class SFS:
        def search(self,classifier,estimator,training_file):
                d=read_data(training_file)
                ind=range(1,71)
                feature=[]
                tmp=self.ADD_one_feature({},70,d,classifier,estimator)
                feature+=tmp
                for i in range(2,6):
                    tmp=[]
                    for j in range(len(feature)):
                        tmp+=self.ADD_one_feature(feature[j],70,d,classifier,estimator)
                    tt=[]
                    optimal=0
                    for j in tmp:
                        if len(j['o'])==i:
                            tt.append(j)
                            if j['o'][i-1]>optimal:
                                optimal=j['o'][i-1]
                    ttt=[]
                    for j in tt:
                        if j['o'][i-1]==optimal:
                            ttt.append(j)
                    feature=ttt
                return feature
```

```
def ADD_one_feature(self,pre_feature,N,d,classifier,estimator):
        ind=range(1,N+1)
        pf=[]
        po=[]
        if len(pre_feature)!=0:
            pf=pre_feature['f']
            po=pre_feature['o']
            for i in pf:
                ind.remove(i)
        C=[]
        for i in ind:
                f=pf+[i]
                data=read_col(f,d)
                label=read_col_label(71,d)
                corr=estimator.estimate(classifier,data,label)
                C.append({'feature':i,'corr':corr})
        s_C=sorted(C,key=itemgetter('corr'))
        optimal=s_C[len(s_C)-1]['corr']
        feature=[]
        for i in s_C:
                if i['corr']==optimal:
                    feature.append({'f':pf+[i['feature']],'o':po+[optimal]})
        return feature
```

Other codes:

init.py (used for data organization)

```
from itertools import combinations
from numpy import matrix
def read_data(file_name):
        f=open(file_name)
        data=[]
        h=f.readline()
        for line in f:
                tmp=line[0:len(line)-1].split("\t")
                floats=[float(x) for x in tmp]
                data.append(floats)
        return matrix(data)
def read_col(col_list,matrix):
        return matrix[:,col_list].tolist()

def read_col_label(col,matrix):
        return matrix[:,col].transpose().tolist()[0]
```

main.py (main file for exhaustive search)

```
#!/usr/bin/env python
from feature_sel import *
from KNN import *
from err_est import *
from SVM import *
from dlda import *
from testing import *
Sfs=SFS()
Ex=exhaustive()
Knn=KNN(3)
Svm=SVM()
Dlda=DLDA()
Loo=LOO()
resub=Resub()
```

```
training_file='../Data/Training_Data.txt'
testing_file='../Data/Testing_Data.txt'


testing=read_data(testing_file)
training=read_data(training_file)
testing_l=read_col_label(71,testing)
training_l=read_col_label(71,training)


EX_LOO_KNN=Ex.search(Knn,Loo,training_file)
#print EX_LOO_KNN
for i in EX_LOO_KNN:
    f=i['feature']
    e=i['corr']
    s=final_test(testing,testing_l,training,training_l,f,Knn)
    s='EX\tLOO\tKNN\t'+s+'\t'+str(e)+'\t'+str(len(f))
    print s
EX_RESUB_KNN=Ex.search(Knn,resub,training_file)
#print EX_RESUB_KNN
for i in EX_RESUB_KNN:
    f=i['feature']
    e=i['corr']
    s=final_test(testing,testing_l,training,training_l,f,Knn)
    s='EX\tRESUB\tKNN\t'+s+'\t'+str(e)+'\t'+str(len(f))
    print s

EX_LOO_SVM=Ex.search(Svm,Loo,training_file)
#print EX_LOO_SVM
for i in EX_LOO_SVM:
    f=i['feature']
    e=i['corr']
    s=final_test(testing,testing_l,training,training_l,f,Svm)
    s='EX\tLOO\tSVM\t'+s+'\t'+str(e)+'\t'+str(len(f))
    print s

EX_RESUB_SVM=Ex.search(Svm,resub,training_file)
#print EX_RESUB_SVM
for i in EX_RESUB_SVM:
    f=i['feature']
    e=i['corr']
    s=final_test(testing,testing_l,training,training_l,f,Svm)
    s='EX\tRESUB\tSVM\t'+s+'\t'+str(e)+'\t'+str(len(f))
    print s

EX_LOO_DLDA=Ex.search(Dlda,Loo,training_file)
#print EX_LOO_DLDA
for i in EX_LOO_DLDA:
    f=i['feature']
    e=i['corr']
    s=final_test(testing,testing_l,training,training_l,f,Dlda)
    s='EX\tLOO\tDLDA\t'+s+'\t'+str(e)+'\t'+str(len(f))
    print s

EX_RESUB_DLDA=Ex.search(Dlda,resub,training_file)
#print EX_RESUB_DLDA
for i in EX_RESUB_DLDA:
    f=i['feature']
    e=i['corr']
    s=final_test(testing,testing_l,training,training_l,f,Dlda)
    s='EX\tRESUB\tDLDA\t'+s+'\t'+str(e)+'\t'+str(len(f))
    print s
```

main2.py (for sequential forward search)

```
#!/usr/bin/env python
from feature_sel import *
from KNN import *
from err_est import *
from SVM import *
from dlda import *
from testing import *
Sfs=SFS()
Ex=exhaustive()
Knn=KNN(3)
Svm=SVM()
Dlda=DLDA()
Loo=LOO()
resub=Resub()
training_file='../Data/Training_Data.txt'
testing_file='../Data/Testing_Data.txt'

testing=read_data(testing_file)
training=read_data(training_file)
testing_l=read_col_label(71,testing)
training_l=read_col_label(71,training)


SFS_LOO_SVM=Sfs.search(Svm,Loo,training_file)
#print SFS_LOO_SVM
for i in SFS_LOO_SVM:
    feature=i['f']
    est_err=i['o']
    for j in range(1,len(est_err)+1):
        f=feature[0:j]
        e=est_err[j-1]
        s=final_test(testing,testing_l,training,training_l,f,Svm)
        s='SFS\tLOO\tSVM\t'+s+'\t'+str(e)+'\t'+str(len(f))
        print s
SFS_RESUB_SVM=Sfs.search(Svm,resub,training_file)
#print SFS_RESUB_SVM
for i in SFS_RESUB_SVM:
    feature=i['f']
    est_err=i['o']
    for j in range(1,len(est_err)+1):
        f=feature[0:j]
        e=est_err[j-1]
        s=final_test(testing,testing_l,training,training_l,f,Svm)
        s='SFS\tRESUB\tSVM\t'+s+'\t'+str(e)+'\t'+str(len(f))
        print s
SFS_LOO_KNN=Sfs.search(Knn,Loo,training_file)
#print SFS_LOO_KNN
for i in SFS_LOO_KNN:
    feature=i['f']
    est_err=i['o']
    for j in range(1,len(est_err)+1):
        f=feature[0:j]
        e=est_err[j-1]
        s=final_test(testing,testing_l,training,training_l,f,Knn)
        s='SFS\tLOO\tKNN\t'+s+'\t'+str(e)+'\t'+str(len(f))
        print s
SFS_RESUB_KNN=Sfs.search(Knn,resub,training_file)
#print SFS_RESUB_KNN
for i in SFS_RESUB_KNN:
    feature=i['f']
```

```
    est_err=i['o']
    for j in range(1,len(est_err)+1):
        f=feature[0:j]
        e=est_err[j-1]
        s=final_test(testing,testing_l,training,training_l,f,Knn)
        s='SFS\tRESUB\tKNN\t'+s+'\t'+str(e)+'\t'+str(len(f))
        print s
SFS_LOO_DLDA=Sfs.search(Dlda,Loo,training_file)
#print SFS_LOO_DLDA
for i in SFS_LOO_DLDA:
    feature=i['f']
    est_err=i['o']
    for j in range(1,len(est_err)+1):
        f=feature[0:j]
        e=est_err[j-1]
        s=final_test(testing,testing_l,training,training_l,f,Dlda)
        s='SFS\tLOO\tDLDA\t'+s+'\t'+str(e)+'\t'+str(len(f))
        print s
SFS_RESUB_DLDA=Sfs.search(Dlda,resub,training_file)
#print SFS_RESUB_DLDA
for i in SFS_RESUB_DLDA:
    feature=i['f']
    est_err=i['o']
    for j in range(1,len(est_err)+1):
        f=feature[0:j]
        e=est_err[j-1]
        s=final_test(testing,testing_l,training,training_l,f,Dlda)
        s='SFS\tRESUB\tDLDA\t'+s+'\t'+str(e)+'\t'+str(len(f))
        print s
```

Testing.py (used for data testing after classier is generated with selected features)

```
from init import *
def
final_test(testing_data,testing_label,training_data,training_label,feature,classifier):
    t_d=read_col(feature,testing_data)
    d=read_col(feature,training_data)
    classifier.training(d,training_label)
    r=classifier.testing(t_d,testing_label)
    corr=float(sum(r))/float(len(testing_label))
    s=''
    for i in feature:
        s=s+str(i)+','
    x=sorted(feature)
    ss=''
    for i in x:
        ss=ss+str(i)+','
    return s+"\t"+ss+'\t'+str(corr)
```