**Materials Informatics – Fall 2017**

**Computer Project 2 – Solutions**

**Due on: Oct 24 2017 11:59pm**

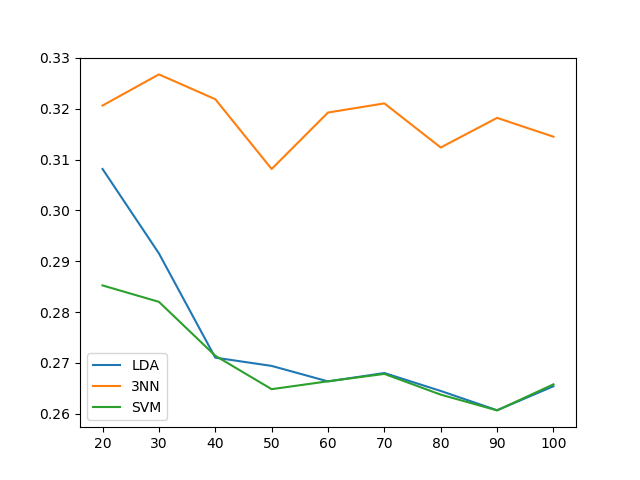
Jianfeng Song

UIN:426009910

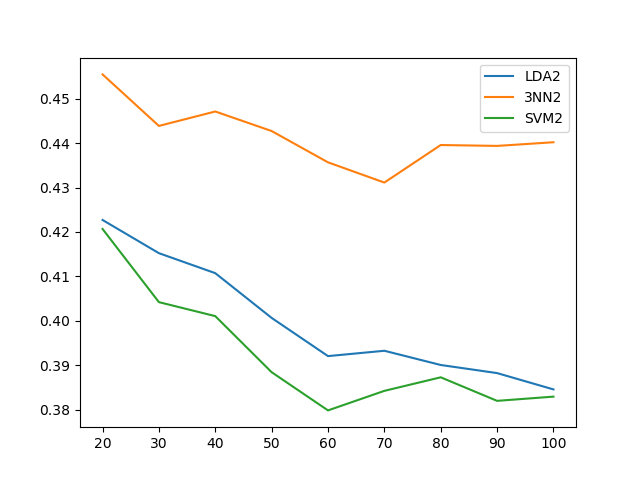
[Jsong26@tamu.edu](mailto:Jsong26@tamu.edu)

Assignment 1

1. The picture below is when σ=1

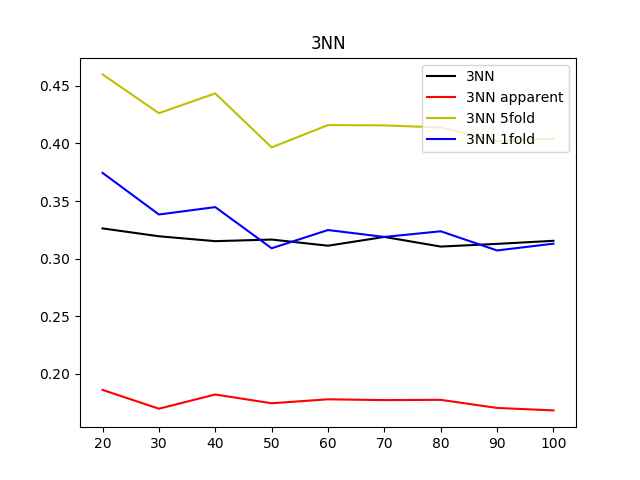


The picture below is when σ=2

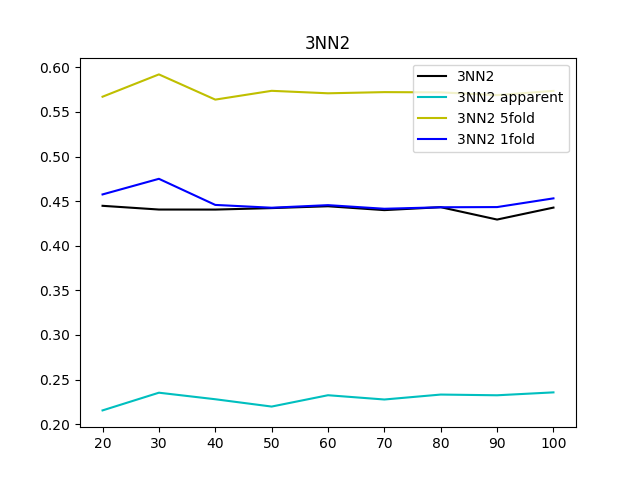


From two pictures above, we can find that SVM is better than LDA, and 3NN is the worst one this result does not depend on the value of σ. When σ is getting large, the corresponding classification error is also increase. When the number of training set increase, the corresponding classification error get decreased

b).

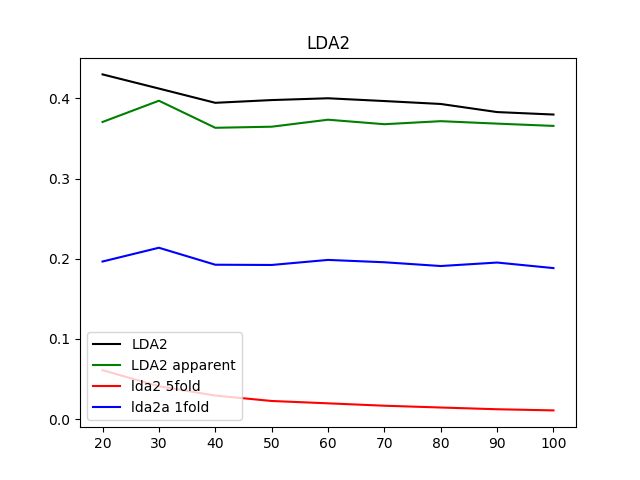


This is 3NN classifier with σ=1

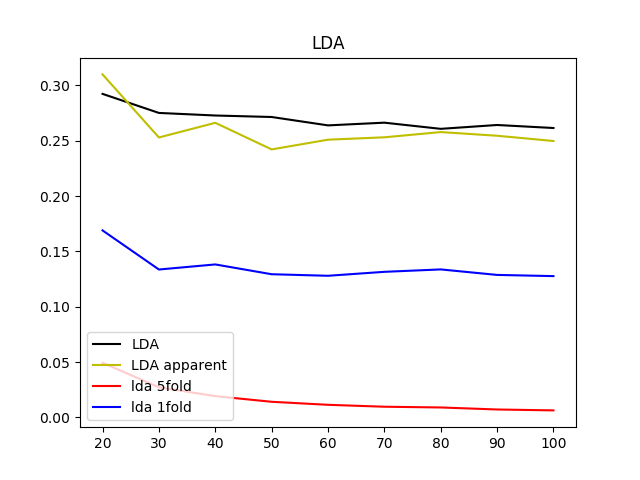


This is 3NN classifier with σ=2

From the pictures above, we can find that the test set error is much close to Leave\_one\_out error estimates for 3NN with both σ=2 and σ=1. So for 3NN I will choose leave-one-out error estimator.

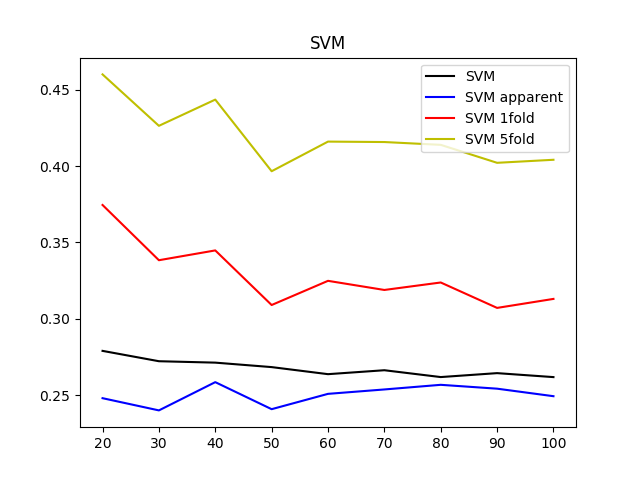


The picture above is LDA classifier with σ=2

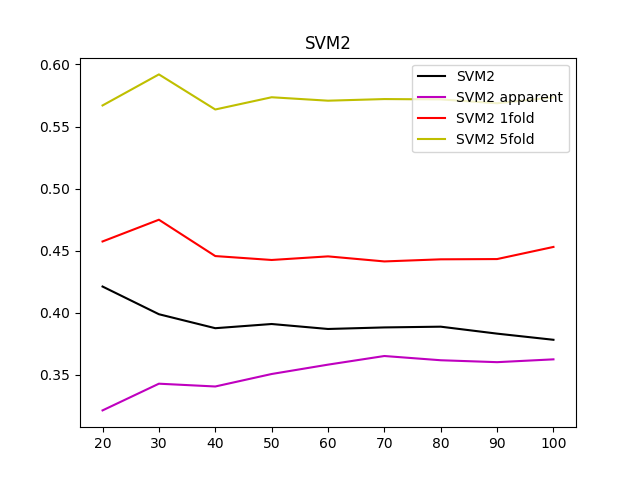


The picture above is LDA classifier with σ=1

From tow above pictures we can find that the test set error is much close to apparent error error estimates for LDA with σ=1 and σ=2. So for LDA I will choose apparent error estimator.



The picture above is SVM with σ=1



The picture above is SVM with σ=2

From tow above pictures we can find that the test set error is much close to apparent error error estimates for SVM with σ=1 and σ=2. So for SVM I will choose apparent error estimator.

Assignment 2

|  |  |  |
| --- | --- | --- |
| Ehaustive Search LDA | error estimate | test-set estimate |
| Fe | 0.12 | 0.1428 |
| C,Fe | 0.04 | 0.1224 |
| C,Ni,Fe | 0.04 | 0.0612 |
| C,N,Fe,Mn | 0.04 | 0.1122 |
| N,Ni,Fe,Si,Cr | 0 | 0.1632 |
|  |  |  |
| Ehaustive Search 3NN | error estimate | test-set estimate |
| Mn | 0.04 | 0.2551 |
| C,Mn | 0.04 | 0.2346 |
| C,N,Mn | 0.04 | 0.2346 |
| C,N,Ni,Fe | 0.04 | 0.0612 |
| C,N,Ni,Fe,Mn | 0.04 | 0.0612 |
|  |  |  |
| Sequential Forward Search LDA | error estimate | test-set estimate |
| Fe | 0.12 | 0.1428 |
| Fe,C | 0.04 | 0.1224 |
| Fe,C,Ni | 0.04 | 0.0612 |
| Fe,C,Ni,Mn | 0.04 | 0.0612 |
| Fe,C,Ni,Mn,N | 0.04 | 0.0918 |
|  |  |  |
| Sequential Forward Search 3NN | error estimate | test-set estimate |
| Mn | 0.04 | 0.2551 |
| Mn,C | 0.04 | 0.2346 |
| Mn,C,N | 0.04 | 0.2346 |
| Mn,C,N,Si | 0.04 | 0.2244 |
| Mn,C,N,Si,Ni | 0.04 | 0.0918 |
|  |  |  |

**How do you compare the results against each other and against the results obtained with the simple filter feature selection used in Project 1?**

Compare with solution in project 1, I can find that two best individual p values (Ni, Fe) do not form the best apparent error estimator (C, Fe). And from the result, we can find that the variables picked up by Ehaustive Search have minimum true error for both LDA and 3NN, but it will take much longer to finish the computing. On the other hand, if we do not need low test error, sequential forward search can always be a good choice.

**How do you compare the error estimators and feature selection methods used based on the variable sets found and the estimates of the true error?**

For LDA, I will use Sequential Forward Search to find my error estimator, the minimum true error appears when we have only three variables(Fe,C,Ni). Since the Sequential Forward Search method use less energy than Ehaustive Search, so I will use it for LDA.

For 3NN, I will use Ehaustive Search to find my error estimator, if we need to have small true error, but this method will cast more energy. If we do not need very high accuracy Sequential Forward Search will be my first choice.

**How do you think the results might change if there were more training points available?**

If we have more training points, the true error from our error estimator will be more unbiased this will fit both exhaustive search and sequential forward search, and it will make exhaustive search much longer.

Code

Assignment 1

a).

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 20 12:13:35 2017

@author: jianfengsong

"""

import numpy as np

import matplotlib.pyplot as plt

import scipy.stats as ns

import math

from scipy.linalg import det

from numpy.linalg import inv

from sklearn.neighbors import NearestNeighbors

from sklearn.neighbors import KNeighborsClassifier

from sklearn import svm

p0=np.array([[1,0.2],

[0.2,1]])

p1=np.array([[1,0.2],

[0.2,1]])

pro\_p0=1/2

pro\_p1=1/2

u0=np.array([0,0])

u1=np.array([1,1])

LDA\_error\_set=list()

N3N\_error\_set=list()

LDA\_err\_set=list()

sample\_sizes=np.arange(20,101,10)

large\_number=1000

total\_train\_set=[]

total\_test\_set=[]

class function:

def firsttime(samplesize):

x1=np.random.multivariate\_normal(u1,p1,int(samplesize/2))

x0=np.random.multivariate\_normal(u0,p0,int(samplesize/2))

return x1,x0

def secondtime(samplesize):

x1=np.random.multivariate\_normal(u1,4\*p1,int(samplesize/2))

x0=np.random.multivariate\_normal(u0,4\*p0,int(samplesize/2))

return x1,x0

def LDA\_error(samplesize,x1,x0):#x1 is the trainning set with mean 1, p is coveriance matrix given in the question, u0 is true mean[0,0]

sumx1=0

sumx0=0

for a in x1:

sumx1=a+sumx1

for b in x0:

sumx0=b+sumx0

mean1=sumx1/samplesize\*2

mean0=sumx0/samplesize\*2

cov=(1/(samplesize-2))\*(np.matrix((x1-mean1)).T\*np.matrix((x1-mean1))+np.matrix((x0-mean0)).T\*np.matrix((x0-mean0)))

an=np.matrix(inv(cov))\*np.matrix((mean1-mean0)).T

bn=(-1/2)\*np.matrix((mean1-mean0))\*np.matrix(inv(cov))\*np.matrix(mean1+mean0).T

varx0=(np.dot(an.T,u0)+bn)/np.sqrt(np.dot(np.dot(an.T,cov),an))

varx1=(np.dot(an.T,u1)+bn)/np.sqrt(np.dot(np.dot(an.T,cov),an))

LDA\_err=1/2\*(ns.norm.cdf(varx0)+ns.norm.cdf(-varx1))

return LDA\_err,an,bn,cov

def Cla\_error\_LDA(an,bn,test1,test0):

clas\_x1\_y=-bn/an[1]-an[0]\*test1[:,0]/an[1]

clas\_x0\_y=-bn/an[1]-an[0]\*test0[:,0]/an[1]

error\_time=0

for t in range(200):

if test1[t,1] < clas\_x1\_y[0,t]:

error\_time=error\_time+1

if test0[t,1] > clas\_x0\_y[0,t]:

error\_time=error\_time+1

return error\_time/400

def data (x1\_train,x0\_train,test1,test0):

train=list()

tar=list()

test=list()

for a in range(len(x1\_train)):

train.append(x1\_train[a])

for a in range(len(x0\_train)):

train.append(x0\_train[a])

for a in range(len(test1)):

test.append(test1[a])

for a in range(len(test0)):

test.append(test0[a])

for a in range(int(sample\_size/2)):

tar.append(1)

for a in range(int(sample\_size/2)):

tar.append(0)

return train,test,tar

def determind(data\_set):

number\_of\_wrong=0

for a in range(0,int(len(data\_set)/2)):

if data\_set[a]<1:

number\_of\_wrong+=1

for a in range(int(len(data\_set)/2),len(data\_set)):

if data\_set[a]>0:

number\_of\_wrong+=1

error\_rate=number\_of\_wrong/len(data\_set)

return error\_rate

#main function##########################################################################

for z in range(2):

error\_perc\_set=list()

nn3\_error\_set=list()

svm\_error\_set=list()

for sample\_size in sample\_sizes:

LDA\_errs=0

error\_percs=0

nn3error=0

nn3errors=0

svmerror=0

svmerrors=0

for a in range(0,100):

nn3mis=0

svmmis=0

if z==0:

x1\_train,x0\_train=function.firsttime(sample\_size)

test1,test0=function.firsttime(400)

if z==1:

x1\_train,x0\_train=function.secondtime(sample\_size)

test1,test0=function.secondtime(400)

#LDA

LDA\_err\_1,an,bn,cov=function.LDA\_error(sample\_size,x1\_train,x0\_train)

error\_perc=function.Cla\_error\_LDA(an,bn,test1,test0)

LDA\_errs+=LDA\_err\_1[0][0]

error\_percs+=error\_perc

#3NN

train,test,tar=function.data(x1\_train,x0\_train,test1,test0)

nn3=KNeighborsClassifier(n\_neighbors=3)

nn3.fit(train,tar)

nn3\_clas=nn3.predict(test)

nn3error=function.determind(nn3\_clas)

nn3errors+=nn3error

#SVM

svm\_cla=svm.LinearSVC()

svm\_cla.fit(train,tar)

svm\_cla\_set=svm\_cla.predict(test)

svmerror=function.determind(svm\_cla\_set)

svmerrors+=svmerror

#error set

svm\_error=svmerrors/100

svm\_error\_set.append(svm\_error)

nn3\_error=nn3errors/100

nn3\_error\_set.append(nn3\_error)

error\_perc=error\_percs/100

error\_perc\_set.append(error\_perc)

if z==0:

plt.figure(1)

plt.plot(sample\_sizes,error\_perc\_set,label='LDA')

plt.legend()

plt.figure(1)

plt.plot(sample\_sizes,nn3\_error\_set,label='3NN')

plt.legend()

plt.figure(1)

plt.plot(sample\_sizes,svm\_error\_set,label='SVM')

plt.legend()

if z==1:

plt.figure(2)

plt.plot(sample\_sizes,error\_perc\_set,label='LDA2')

plt.legend()

plt.figure(2)

plt.plot(sample\_sizes,nn3\_error\_set,label='3NN2')

plt.legend()

plt.figure(2)

plt.plot(sample\_sizes,svm\_error\_set,label='SVM2')

plt.legend()

plt.show()

b).

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 20 12:13:35 2017

@author: jianfengsong

"""

import numpy as np

import matplotlib.pyplot as plt

from numpy.linalg import inv

from sklearn.neighbors import KNeighborsClassifier

from sklearn import svm

import scipy.stats as ns

from sklearn.model\_selection import KFold

from sklearn.model\_selection import LeaveOneOut

p0=np.array([[1,0.2],

[0.2,1]])

p1=np.array([[1,0.2],

[0.2,1]])

pro\_p0=1/2

pro\_p1=1/2

u0=np.array([0,0])

u1=np.array([1,1])

LDA\_error\_set=list()

N3N\_error\_set=list()

LDA\_err\_set=list()

sample\_sizes=np.arange(20,101,10)

large\_number=1000

total\_train\_set=[]

total\_test\_set=[]

class function:

def firsttime(samplesize):

x1=np.random.multivariate\_normal(u1,p1,int(samplesize/2))

x0=np.random.multivariate\_normal(u0,p0,int(samplesize/2))

return x1,x0

def secondtime(samplesize):

x1=np.random.multivariate\_normal(u1,4\*p1,int(samplesize/2))

x0=np.random.multivariate\_normal(u0,4\*p0,int(samplesize/2))

return x1,x0

def LDA\_error(samplesize,x1,x0):#x1 is the trainning set with mean 1, p is coveriance matrix given in the question, u0 is true mean[0,0]

sumx1=0

sumx0=0

for a in x1:

sumx1=a+sumx1

for b in x0:

sumx0=b+sumx0

mean1=sumx1/samplesize\*2

mean0=sumx0/samplesize\*2

cov=(1/(samplesize-2))\*(np.matrix((x1-mean1)).T\*np.matrix((x1-mean1))+np.matrix((x0-mean0)).T\*np.matrix((x0-mean0)))

an=np.matrix(inv(cov))\*np.matrix((mean1-mean0)).T

bn=(-1/2)\*np.matrix((mean1-mean0))\*np.matrix(inv(cov))\*np.matrix(mean1+mean0).T

varx0=(np.dot(an.T,u0)+bn)/np.sqrt(np.dot(np.dot(an.T,cov),an))

varx1=(np.dot(an.T,u1)+bn)/np.sqrt(np.dot(np.dot(an.T,cov),an))

LDA\_err=1/2\*(ns.norm.cdf(varx0)+ns.norm.cdf(-varx1))

return LDA\_err,an,bn,cov

def Cla\_error\_LDA(an,bn,test1,test0):

if len(test1)!=0:

clas\_x1\_y=-bn/an[1]-an[0]\*test1[:,0]/an[1]

if len(test0)!=0:

clas\_x0\_y=-bn/an[1]-an[0]\*test0[:,0]/an[1]

error\_time=0

for t in range(len(test1)):

if len(test1)!=0:

if test1[t,1] < clas\_x1\_y[0,t]:

error\_time=error\_time+1

if len(test0)!=0:

if test0[t,1] > clas\_x0\_y[0,t]:

error\_time=error\_time+1

return error\_time/(len(test1)+len(test0))

def data (x1\_train,x0\_train,test1,test0):

train=list()

tar=list()

test=list()

for a in range(len(x1\_train)):

train.append(x1\_train[a])

for a in range(len(x0\_train)):

train.append(x0\_train[a])

for a in range(len(test1)):

test.append(test1[a])

for a in range(len(test0)):

test.append(test0[a])

for a in range(int(sample\_size/2)):

tar.append(1)

for a in range(int(sample\_size/2)):

tar.append(0)

return train,test,tar

def determind(data\_set):

number\_of\_wrong=0

for a in range(0,int(len(data\_set)/2)):

if data\_set[a]<1:

number\_of\_wrong+=1

for a in range(int(len(data\_set)/2),len(data\_set)):

if data\_set[a]>0:

number\_of\_wrong+=1

error\_rate=number\_of\_wrong/len(data\_set)

return error\_rate

def kfold\_3nn(train1,tar1,num,sample\_size):

train=np.asarray(train1)

tar=np.asarray(tar1)

numberofwrong=0

wrong\_pro=0

time=0

if num!=1:

kf=KFold(n\_splits=num)

kf.get\_n\_splits(train)

for train\_index, test\_index in kf.split(train):

time+=1

x\_train, x\_test = train[train\_index], train[test\_index]

y\_train, y\_test = tar[train\_index], tar[test\_index]

nn3=KNeighborsClassifier(n\_neighbors=3)

nn3.fit(x\_train,y\_train)

nn3\_clas=nn3.predict(x\_test)

for a in range(len(nn3\_clas)):

if nn3\_clas[a] != y\_test[a]:

numberofwrong+=1

wrong\_pro=numberofwrong/(sample\_size)

else:

loo = LeaveOneOut()

loo.get\_n\_splits(train)

for train\_index, test\_index in loo.split(train):

x\_train, x\_test = train[train\_index], train[test\_index]

y\_train, y\_test = tar[train\_index], tar[test\_index]

nn3=KNeighborsClassifier(n\_neighbors=3)

nn3.fit(x\_train,y\_train)

nn3\_clas=nn3.predict(x\_test)

for a in range(len(nn3\_clas)):

if nn3\_clas[a] != y\_test[a]:

numberofwrong+=1

wrong\_pro=numberofwrong/(sample\_size)

return wrong\_pro

def kfold\_SVM(train1,tar1,n,sample\_size):

train=np.asarray(train1)

tar=np.asarray(tar1)

numberofwrong=0

wrong\_pro=0

if n!=1:

kf=KFold(n\_splits=n)

kf.get\_n\_splits(train)

for train\_index, test\_index in kf.split(train):

x\_train, x\_test = train[train\_index], train[test\_index]

y\_train, y\_test = tar[train\_index], tar[test\_index]

svm\_cla=svm.LinearSVC()

svm\_cla.fit(x\_train,y\_train)

svm\_cla\_set=svm\_cla.predict(x\_test)

for a in range(len(svm\_cla\_set)):

if svm\_cla\_set[a]!=y\_test[a]:

numberofwrong+=1

wrong\_pro=numberofwrong/(sample\_size)

else:

loo = LeaveOneOut()

loo.get\_n\_splits(train)

for train\_index, test\_index in loo.split(train):

x\_train, x\_test = train[train\_index], train[test\_index]

y\_train, y\_test = tar[train\_index], tar[test\_index]

svm\_cla=svm.LinearSVC()

svm\_cla.fit(x\_train,y\_train)

svm\_cla\_set=svm\_cla.predict(x\_test)

for a in range(len(svm\_cla\_set)):

if svm\_cla\_set[a]!=y\_test[a]:

numberofwrong+=1

wrong\_pro=numberofwrong/(sample\_size)

return wrong\_pro

def kfold\_LDA(train1,tar1,num,sample\_size):

train=np.asarray(train1)

tar=np.asarray(tar1)

numberofwrong=0

wrong\_pro=0

if num!=1:

kf=KFold(n\_splits=num)

kf.get\_n\_splits(train)

for train\_index, test\_index in kf.split(train):

x\_train, x\_test = train[train\_index], train[test\_index]

y\_train, y\_test = tar[train\_index], tar[test\_index]

x1train,x0train,x1test,x0test=function.kfold\_LDA\_data(train,tar,x\_train,x\_test,y\_train,y\_test)

LDA,an,bn,cov=function.LDA\_error(sample\_size,x1train,x0train)

numberofwrong+=function.Cla\_error\_LDA(an,bn,x1test,x0test)

wrong\_pro=numberofwrong/sample\_size

else:

loo = LeaveOneOut()

loo.get\_n\_splits(train)

for train\_index, test\_index in loo.split(train):

x\_train, x\_test = train[train\_index], train[test\_index]

y\_train, y\_test = tar[train\_index], tar[test\_index]

x1train,x0train,x1test,x0test=function.kfold\_LDA\_data(train,tar,x\_train,x\_test,y\_train,y\_test)

LDA,an,bn,cov=function.LDA\_error(sample\_size,x1train,x0train)

numberofwrong+=function.Cla\_error\_LDA(an,bn,x1test,x0test)

wrong\_pro=numberofwrong/sample\_size

return wrong\_pro

def kfold\_LDA\_data(train,tar,x\_train,x\_test,y\_train,y\_test):

x1train=[]

x1test=[]

x0train=[]

x0test=[]

for a in range(len(y\_train)):

if y\_train[a]==1:

x1train.append(x\_train[a])

else:

x0train.append(x\_train[a])

for a in range(len(y\_test)):

if y\_test[a]==1:

x1test.append(x\_test[a])

else:

x0test.append(x\_test[a])

x1train1=np.asarray(x1train)

x0train0=np.asarray(x0train)

x1test1=np.asarray(x1test)

x0test0=np.asarray(x0test)

return x1train1,x0train0,x1test1,x0test0

#main function##########################################################################

for z in range(2):

error\_perc\_set=list()

nn3\_error\_set=list()

nn3\_5fold\_error\_set=list()

nn3\_1fold\_error\_set=list()

svm\_error\_set=list()

svm\_5fold\_error\_set=list()

svm\_1fold\_error\_set=list()

lda\_error\_set=list()

lda\_5fold\_error\_set=list()

lda\_1fold\_error\_set=list()

for sample\_size in sample\_sizes:

LDA\_errs=0

error\_percs=0

nn3error=0

nn3errors=0

nn3\_1fold=0

nn3\_5fold=0

nn3\_5fold\_errors=0

nn3\_1fold\_errors=0

svmerror=0

svmerrors=0

svm\_1fold=0

svm\_5fold=0

svm\_5fold\_errors=0

svm\_1fold\_errors=0

lda\_1fold=0

lda\_5fold=0

lda\_5fold\_errors=0

lda\_1fold\_errors=0

for a in range(0,100):

nn3mis=0

svmmis=0

if z==0:

x1\_train,x0\_train=function.firsttime(sample\_size)

test1,test0=function.firsttime(400)

if z==1:

x1\_train,x0\_train=function.secondtime(sample\_size)

test1,test0=function.secondtime(400)

train,test,tar=function.data(x1\_train,x0\_train,test1,test0)

#LDA

LDA\_err\_1,an,bn,cov=function.LDA\_error(sample\_size,x1\_train,x0\_train)

error\_perc=function.Cla\_error\_LDA(an,bn,x1\_train,x0\_train)

LDA\_errs+=LDA\_err\_1[0][0]

error\_percs+=error\_perc

lda\_1fold=function.kfold\_LDA(train,tar,1,sample\_size)

lda\_1fold\_errors+=lda\_1fold

lda\_5fold=function.kfold\_LDA(train,tar,5,sample\_size)

lda\_5fold\_errors+=lda\_5fold

#3NN

nn3=KNeighborsClassifier(n\_neighbors=3)

nn3.fit(train,tar)

nn3\_clas=nn3.predict(train)

nn3error=function.determind(nn3\_clas)

nn3errors+=nn3error

nn3\_1fold=function.kfold\_3nn(train,tar,1,sample\_size)

nn3\_1fold\_errors+=nn3\_1fold

nn3\_5fold=function.kfold\_3nn(train,tar,5,sample\_size)

nn3\_5fold\_errors+=nn3\_5fold

#SVM

svm\_cla=svm.LinearSVC()

svm\_cla.fit(train,tar)

svm\_cla\_set=svm\_cla.predict(train)

svmerror=function.determind(svm\_cla\_set)

svmerrors+=svmerror

svm\_1fold=function.kfold\_SVM(train,tar,1,sample\_size)

svm\_1fold\_errors+=nn3\_1fold

svm\_5fold=function.kfold\_SVM(train,tar,5,sample\_size)

svm\_5fold\_errors+=nn3\_5fold

# print(a\*sample\_size)

#error set

svm\_error=svmerrors/100

svm\_error\_set.append(svm\_error)

svm\_1fold\_error=svm\_1fold\_errors/100

svm\_1fold\_error\_set.append(svm\_1fold\_error)

svm\_5fold\_error=svm\_5fold\_errors/100

svm\_5fold\_error\_set.append(svm\_5fold\_error)

nn3\_error=nn3errors/100

nn3\_error\_set.append(nn3\_error)

nn3\_1fold\_error=nn3\_1fold\_errors/100

nn3\_1fold\_error\_set.append(nn3\_1fold\_error)

nn3\_5fold\_error=nn3\_5fold\_errors/100

nn3\_5fold\_error\_set.append(nn3\_5fold\_error)

error\_perc=error\_percs/100

error\_perc\_set.append(error\_perc)

lda\_1fold\_error=lda\_1fold\_errors/100

lda\_1fold\_error\_set.append(lda\_1fold\_error)

lda\_5fold\_error=lda\_5fold\_errors/100

lda\_5fold\_error\_set.append(lda\_5fold\_error)

if z==0:

plt.figure(1)

plt.title('LDA')

# plt.subplot(311)

plt.plot(sample\_sizes,error\_perc\_set,'y',label='LDA apparent')

plt.plot(sample\_sizes,lda\_5fold\_error\_set,'r',label='lda 5fold')

plt.plot(sample\_sizes,lda\_1fold\_error\_set,'b',label='lda 1fold')

plt.legend()

plt.figure(2)

plt.title('3NN')

# plt.subplot(312)

plt.plot(sample\_sizes,nn3\_error\_set,label='3NN apparent')

plt.plot(sample\_sizes,nn3\_5fold\_error\_set,label='3NN 5fold')

plt.plot(sample\_sizes,nn3\_1fold\_error\_set,label='3NN 1fold')

plt.legend()

plt.figure(3)

plt.title('SVM')

# plt.subplot(313)

plt.plot(sample\_sizes,svm\_error\_set,label='SVM apparent')

plt.plot(sample\_sizes,svm\_1fold\_error\_set,label='SVM 1fold')

plt.plot(sample\_sizes,svm\_5fold\_error\_set,label='SVM 5fold')

plt.legend()

if z==1:

plt.figure(1)

plt.title('LDA2')

# plt.subplot(311)

plt.plot(sample\_sizes,error\_perc\_set,label='LDA2 apparent')

plt.plot(sample\_sizes,lda\_5fold\_error\_set,label='lda2 5fold')

plt.plot(sample\_sizes,lda\_1fold\_error\_set,label='lda2a 1fold')

plt.legend()

plt.figure(2)

plt.title('3NN2')

# plt.subplot(312)

plt.plot(sample\_sizes,nn3\_error\_set,label='3NN2 apparent')

plt.plot(sample\_sizes,nn3\_5fold\_error\_set,label='3NN2 5fold')

plt.plot(sample\_sizes,nn3\_1fold\_error\_set,label='3NN2 1fold')

plt.legend()

plt.figure(3)

plt.title('SVM2')

# plt.subplot(313)

plt.plot(sample\_sizes,svm\_error\_set,label='SVM2 apparent')

plt.plot(sample\_sizes,svm\_1fold\_error\_set,label='SVM2 1fold')

plt.plot(sample\_sizes,svm\_5fold\_error\_set,label='SVM2 5fold')

plt.legend()

plt.show()

Assignment 2

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Sat Oct 21 16:00:46 2017

@author: jianfengsong

"""

import xlrd as xl

import numpy as np

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA

from itertools import combinations

class fun():

def excel\_data(n):

train\_rows\_value=list()

train\_cols\_value=list()

excel=xl.open\_workbook(n)

data\_table=excel.sheet\_by\_index(0)

rows=data\_table.nrows

cols=data\_table.ncols

for a in range(rows):

train\_rows\_value.append(data\_table.row\_values(a))

train\_row=np.asarray(train\_rows\_value)

return train\_row

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

def data (x1\_train,x0\_train,test1,test0):

train=list()

tar=list()

test=list()

test\_tar=list()

for a in range(len(x1\_train)):

train.append(x1\_train[a])

for a in range(len(x0\_train)):

train.append(x0\_train[a])

for a in range(int(len(x1\_train))):

tar.append(1)

for a in range(int(len(x0\_train))):

tar.append(0)

for a in range(len(test1)):

test.append(test1[a])

for a in range(len(test0)):

test.append(test0[a])

for a in range(int(len(test1))):

test\_tar.append(1)

for a in range(int(len(test0))):

test\_tar.append(0)

return train,test,tar,test\_tar

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

def two\_class(data\_set):

data\_high=list()

data\_low=list()

data\_label=list()

for a in range(len(data\_set)):

b=len(data\_set[a])-1

if data\_set[a][b] == 'High':

data\_set[a][b]=1

data\_high.append(data\_set[a])

if data\_set[a][b]=='Low':

data\_set[a][b]=0

data\_low.append(data\_set[a])

if data\_set[a][b] =='SFE':

data\_label.append(data\_set[a])

data\_high1=np.asarray(data\_high)

data\_low1=np.asarray(data\_low)

return data\_high1,data\_low1,data\_label

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

def feature\_sample(data\_set):

feature\_data=[[] for i in range(len(data\_set[1])-1)]

for b in range(len(data\_set[1])-1):

for a in range(len(data\_set)):

feature\_data[b].append(float(data\_set[a][b]))

feature\_data1=np.asarray(feature\_data)

return feature\_data1

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

def get\_feature\_sample(): #find the feature matrix, for example j[0]is all value of 'C'

train\_row=fun.excel\_data('SFE\_Train\_Data.xlsx')

test\_row=fun.excel\_data('SFE\_Test\_Data.xlsx')

train\_high,train\_low,train\_label=fun.two\_class(train\_row)

test\_high,test\_low,test\_label=fun.two\_class(test\_row)

train\_set,test\_set,train\_label,test\_label=fun.data(train\_high,train\_low,test\_high,test\_low)

feature\_col=fun.feature\_sample(train\_set)

test\_col=fun.feature\_sample(test\_set)

# feature\_col1=np.asarray(feature\_col.append(train\_label))

return feature\_col,train\_label,test\_col,test\_label

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

def ehaustive(num):

x=fun.get\_feature\_sample()

selected\_feature\_set=list()

selected\_feature1=combinations(range(7),num)

selected\_feature=np.asarray(list(selected\_feature1))

return selected\_feature

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

def determind(data\_set,tar):

number\_of\_wrong=0

for a in range(len(tar)):

if data\_set[a]!=tar[a]:

number\_of\_wrong+=1

else:

number\_of\_wrong+=0

error\_rate=number\_of\_wrong/len(data\_set)

return error\_rate

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

def NN3\_err(train,tar,test,test\_tar,n):

nn3=KNeighborsClassifier(n\_neighbors=3)

nn3.fit(train,tar)

if n==1:

nn3\_clas=nn3.predict(train)

nn3error=fun.determind(nn3\_clas,tar)

if n==0:

nn3\_clas=nn3.predict(test)

nn3error=fun.determind(nn3\_clas,test\_tar)

return nn3error

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

def LDA\_error(train,tar,test,test\_tar,n):

clf=LDA()

clf.fit(train,tar)

if n ==1:

LDA\_cla=clf.predict(train)

error=fun.determind(LDA\_cla,tar)

if n==0:

LDA\_cla=clf.predict(test)

error=fun.determind(LDA\_cla,test\_tar)

return error

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

############### MAIN() ########################################

feature\_data,feature\_label,test\_set,test\_label=fun.get\_feature\_sample()

min\_ind\_set, min\_err\_set,min\_cla\_err\_set=list(),list(),list()

min3nn\_ind\_set, min3nn\_err\_set,min3nn\_cla\_err\_set=list(),list(),list()

for a in range(1,6):

selected\_feature=fun.ehaustive(a)

min\_err=1

min3nn\_err=1

for b in range(len(selected\_feature)):

selected\_feature\_set1=list()

LDA\_feature1set,LDA\_feature0set,test1,test0=list(),list(),list(),list()

for c in range(len(selected\_feature[b])):

indice=selected\_feature[b][c]

selected\_feature\_set1.append(feature\_data[indice])

selected\_feature\_set=np.asarray(selected\_feature\_set1)

LDA\_feature\_1,LDA\_feature\_0,test\_1,test\_0=list(),list(),list(),list()

for d in range(0,12): #with SFE high

LDA\_feature\_1.append(feature\_data[indice][d])

# test\_1.append(test\_set[indice][d])

for e in range(12,len(feature\_data[c])): #with SFE low

LDA\_feature\_0.append(feature\_data[indice][e])

# test\_0.append(test\_set[indice][e])

for f in range(0,50):

test\_1.append(test\_set[indice][f])

for g in range(50,98):

test\_0.append(test\_set[indice][g])

LDA\_feature1set.append(LDA\_feature\_1)

LDA\_feature0set.append(LDA\_feature\_0)

test1.append(test\_1)

test0.append(test\_0)

#LDA APPARENT

LDA\_feature1\_set=(np.asarray(LDA\_feature1set)).T #as my x1

LDA\_feature0\_set=(np.asarray(LDA\_feature0set)).T #as my x0

test1set=(np.asarray(test1)).T

test0set=(np.asarray(test0)).T

train,test,tar,test\_tar=fun.data(LDA\_feature1\_set,LDA\_feature0\_set,test1set,test0set)

#find min for LDA

LDA\_err=fun.LDA\_error(train,tar,test,test\_tar,1)

LDA\_cla\_err=fun.LDA\_error(train,tar,test,test\_tar,0)

# print(LDA\_cla\_err)

nn3error=fun.NN3\_err(train,tar,test,test\_tar,1)

# print(nn3error,a)

nn3\_cla\_error=fun.NN3\_err(train,tar,test,test\_tar,0)

#LDA

if min\_err>LDA\_err:

min\_err=LDA\_err

min\_ind=b

min\_cla\_err=LDA\_cla\_err

else:

min\_err=min\_err

min\_ind=min\_ind

# 3NN

if min3nn\_err>nn3error:

min3nn\_err=nn3error

min3nn\_ind=b

min3nncla=nn3\_cla\_error

else:

min3nn\_err=min3nn\_err

min3nn\_ind=min3nn\_ind

min\_err\_set.append(min\_err)#min LDA apparent error

min\_ind\_set.append(selected\_feature[min\_ind])#min LDA error index

min\_cla\_err\_set.append(min\_cla\_err)# classification error

min3nn\_err\_set.append(min3nn\_err)#min 3NN apparent error

min3nn\_ind\_set.append(selected\_feature[min3nn\_ind])#min 3nn error index

min3nn\_cla\_err\_set.append(min3nncla)# classification error

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

# SFS for LDA

sfs\_lda=list()

sfs\_lda\_ind=list()

sfs\_ldatrain,sfs\_ldatest=list(),list()

sfs\_lda\_app,sfs\_lda\_cla=list(),list()

sfs\_lda\_app.append(min\_err\_set[0])

sfs\_lda\_cla.append(min\_cla\_err\_set[0])

for a in range(7):

sfs\_lda.append(a)

sfs\_lda.remove(min\_ind\_set[0][0])

sfs\_lda\_ind.append(min\_ind\_set[0][0])

for a in range (4):

sfs\_ldatrain.append(feature\_data[sfs\_lda\_ind[a]])

sfs\_ldatest.append(test\_set[sfs\_lda\_ind[a]])

sfs\_min=1

for b in sfs\_lda:

sfs\_ldatrain.append(feature\_data[b])

sfs\_ldatest.append(test\_set[b])

sfs\_lda\_train=(np.asarray(sfs\_ldatrain)).T

sfs\_lda\_test=(np.asarray(sfs\_ldatest)).T

sfs\_lda\_app\_err=fun.LDA\_error(sfs\_lda\_train,tar,sfs\_lda\_test,test\_label,1)

sfs\_lda\_cla\_err=fun.LDA\_error(sfs\_lda\_train,tar,sfs\_lda\_test,test\_label,0)

if sfs\_min>sfs\_lda\_app\_err:

sfs\_min=sfs\_lda\_app\_err

sfs\_ind=b

min\_cla\_err=sfs\_lda\_cla\_err

else:

sfs\_min=sfs\_min

sfs\_ind=sfs\_ind

sfs\_ldatrain.pop(a+1)

sfs\_ldatest.pop(a+1)

sfs\_lda.remove(sfs\_ind)

sfs\_lda\_app.append(sfs\_min)

sfs\_lda\_cla.append(min\_cla\_err)

sfs\_lda\_ind.append(sfs\_ind)

# SFS 3NN

sfs\_3nn=list()

sfs\_3nn\_ind=list()

sfs\_3nntrain,sfs\_3nntest=list(),list()

sfs\_3nn\_app,sfs\_3nn\_cla=list(),list()

sfs\_3nn\_app.append(min3nn\_err\_set[0])

sfs\_3nn\_cla.append(min3nn\_cla\_err\_set[0])

#sfs\_3nn\_app.append(3)

#sfs\_3nn\_cla.append(min3nn\_cla\_err\_set[0])

for a in range(7):

sfs\_3nn.append(a)

sfs\_3nn.remove(min3nn\_ind\_set[0][0])

sfs\_3nn\_ind.append(min3nn\_ind\_set[0][0])

#sfs\_3nn.remove(3)

#sfs\_3nn\_ind.append(3)

for d in range (4):

sfs\_3nntrain.append(feature\_data[sfs\_3nn\_ind[d]])

sfs\_3nntest.append(test\_set[sfs\_3nn\_ind[d]])

sfs3nn\_min=1

for c in sfs\_3nn:

sfs\_3nntrain.append(feature\_data[c])

sfs\_3nntest.append(test\_set[c])

sfs\_3nn\_train=(np.asarray(sfs\_3nntrain)).T

sfs\_3nn\_test=(np.asarray(sfs\_3nntest)).T

sfs\_3nn\_app\_err=fun.NN3\_err(sfs\_3nn\_train,tar,sfs\_3nn\_test,test\_label,1)

sfs\_3nn\_cla\_err=fun.NN3\_err(sfs\_3nn\_train,tar,sfs\_3nn\_test,test\_label,0)

if sfs3nn\_min>sfs\_3nn\_app\_err:

sfs3nn\_min=sfs\_3nn\_app\_err

sfs3nn\_ind=c

min3nn\_cla\_err=sfs\_3nn\_cla\_err

else:

sfs3nn\_min=sfs3nn\_min

sfs3nn\_ind=sfs3nn\_ind

sfs\_3nntrain.pop(d+1)

sfs\_3nntest.pop(d+1)

sfs\_3nn.remove(sfs3nn\_ind)

sfs\_3nn\_app.append(sfs3nn\_min)

sfs\_3nn\_cla.append(min3nn\_cla\_err)

sfs\_3nn\_ind.append(sfs3nn\_ind)