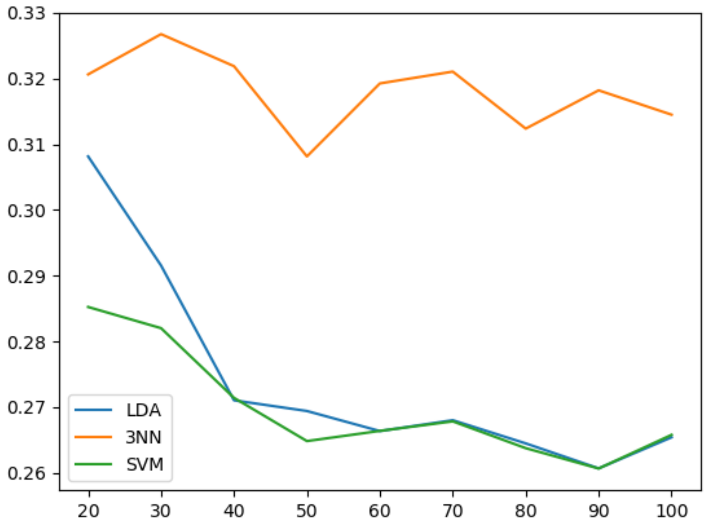
Materials Informatics – Fall 2017
Computer Project 2 – Solutions
Due on: Oct 24 2017 11:59pm

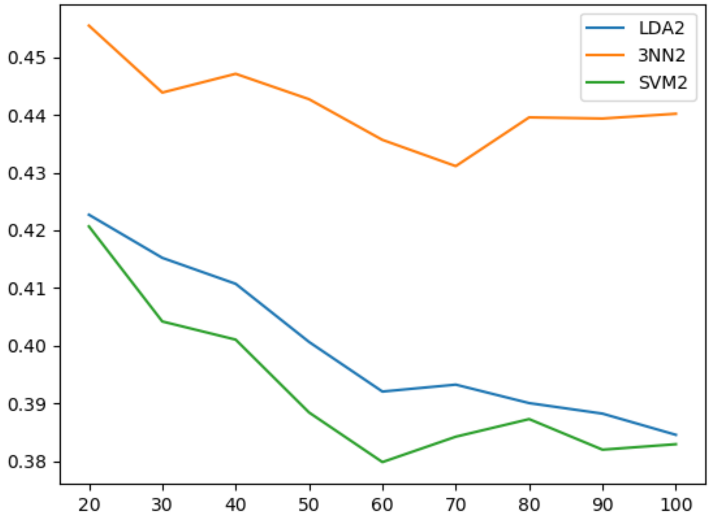
Jianfeng Song
UIN:426009910
Jsong26@tamu.edu

Assignment 1

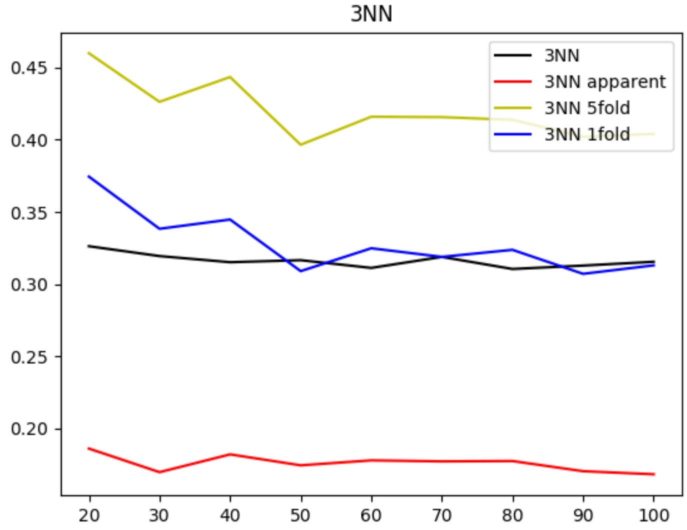
a) The picture below is when σ=1



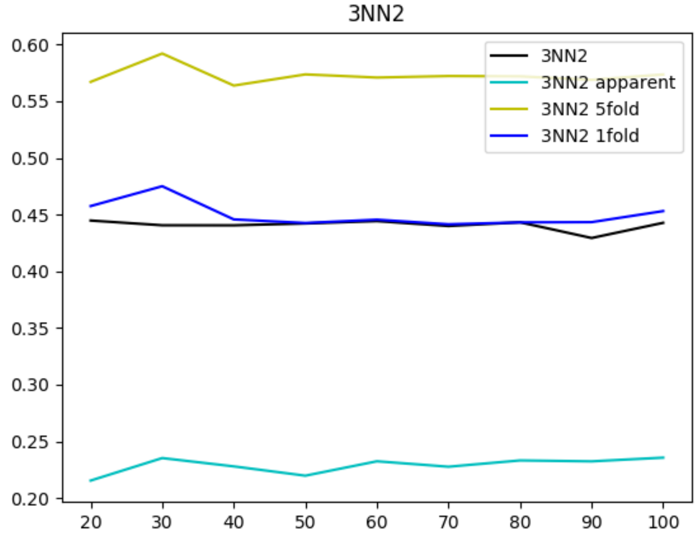
The picture below is when σ=2



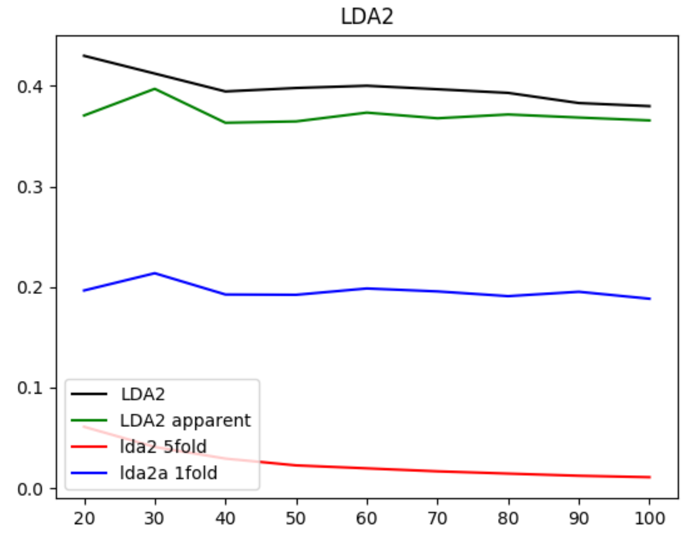
From two pictures above, we can find that SVM is better than LDA, and 3NN is the worst one this result does not depend on the value of σ. When σ is getting large, the corresponding classification error is also increase. When the number of training set increase, the corresponding classification error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



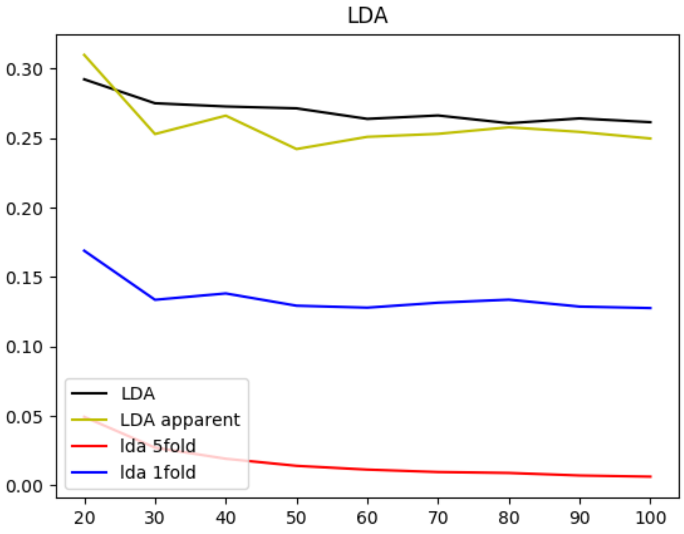
This is 3NN classifier with σ=1



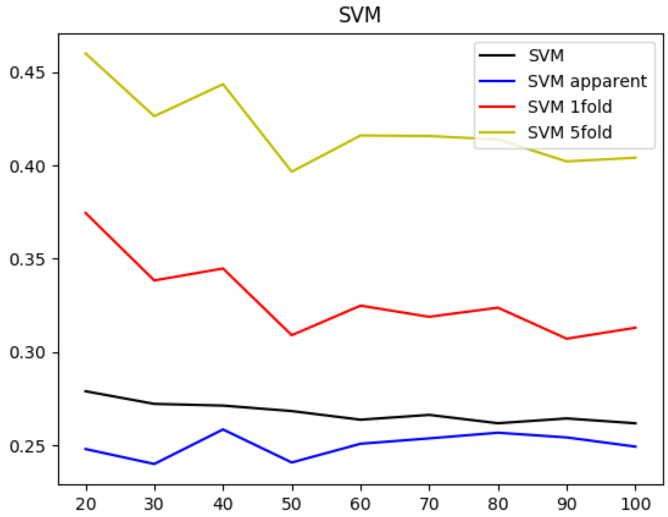
This is 3NN classifier with σ=2
From the pictures above, we can find that the test set error is much close to Leave_one_out error estimates for 3NN with both σ=2 and σ=1. So for 3NN I will choose leave-one-out error estimator.



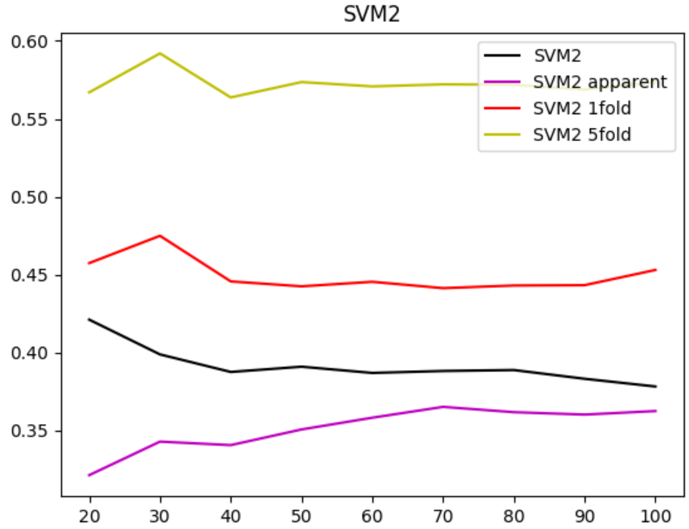
The picture above is LDA classifier with σ=2



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)
```