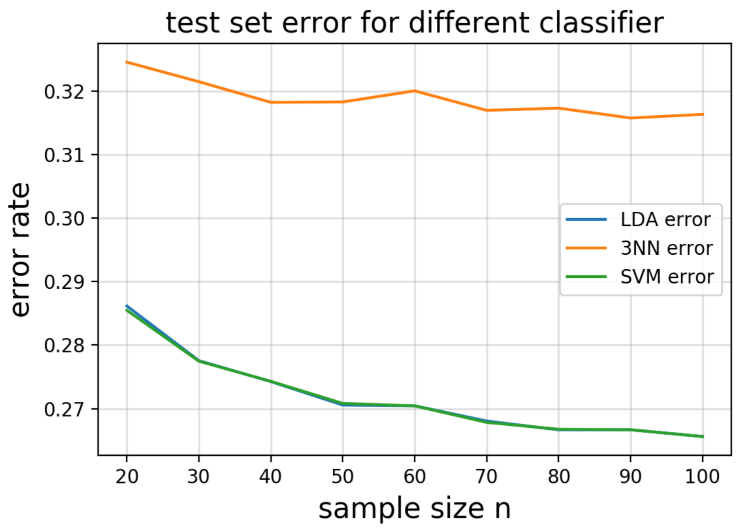
ECEN689-605 Computer Project 2

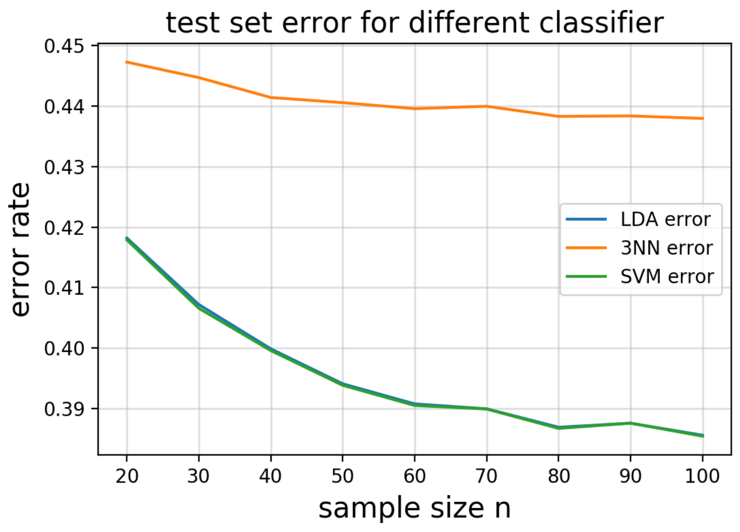
Shutong Jiao

Assignment 1

(a)



(a) $\sigma = 1$



(b) $\sigma = 2$

Figure 1: Average classification errors of the LDA, 3NN, and linear SVM classification rules

From the figure below we could conclude that LDA and SVM classifier have almost the same error rate, while 3NN classifier shows a significantly larger error rate. When $\sigma = 2$ increases from 1 to 2, error rates of all three classifier increases too but the overall pattern is similar to the case when $\sigma = 1$.

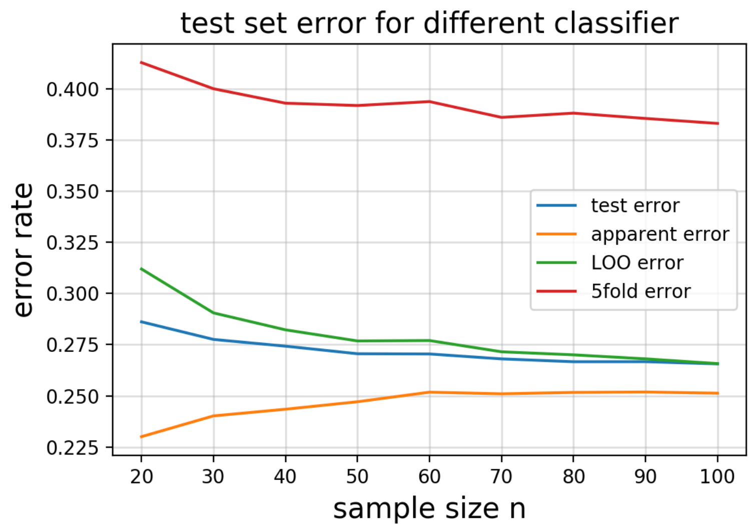
This could be explained by considering the difference between parametric classifier (like LDA) and non-parametric classifier (like 3NN). In the case of LDA, as a parametric classifier, it assumes the train set to be of gaussian distribution and thus expects a linear decision boundary. On the other hand, non-parametric classifier like 3NN makes no assumptions about the shape of the decision boundary and usually the decision boundary they produce will be highly non-linear, which in this case will lead to higher classification error.

For SVM, although it makes no assumptions like 3NN, it also expects a linear decision boundary. Therefore SVM and LDA show very close classification error.

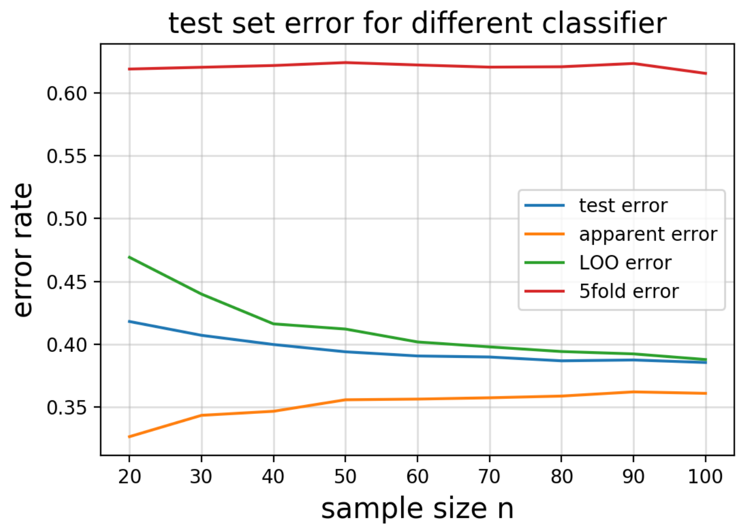
(b)

It can be concluded from the figures below that generally apparent error is optimistically biased estimator while LOO and 5fold error are pessimistically biased. For the absolute value of bias in case of LDA and SVM, LOO is the smallest, followed by apparent error and 5fold is the largest and far more larger than other two. However, for bias in 3NN classifier, apparent error has largest absolute value of bias, followed by 5fold and LOO still has the smallest bias. Besides, LOO could be considered as a special case in KFold in which K equals to sample size n, thus we could expect that when K increases, the error estimates made by KFold will gradually converge to that of LOO.

To conclude, normally I would choose LOO error estimator for all three classification rule if its computational cost is reasonable. For cases when we could not afford the high computation cost of LOO error estimator, I would recommend apparent error for LDA and SVM and 5fold error for 3NN based on the observations from the above figure.

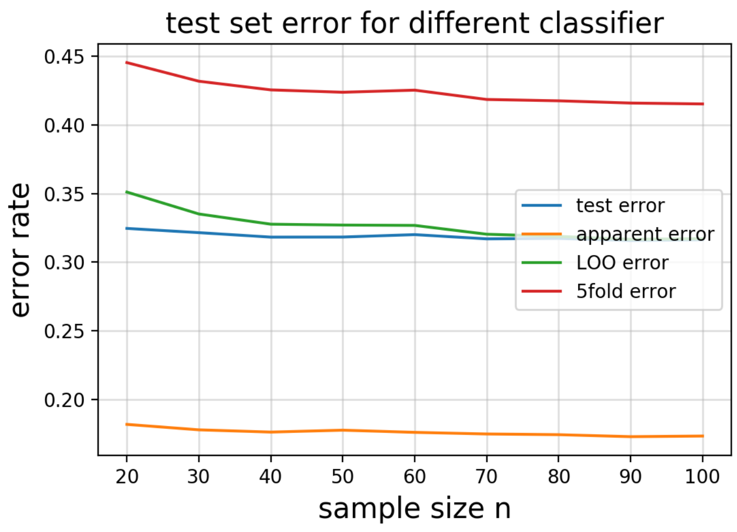


(a) $\sigma = 1$

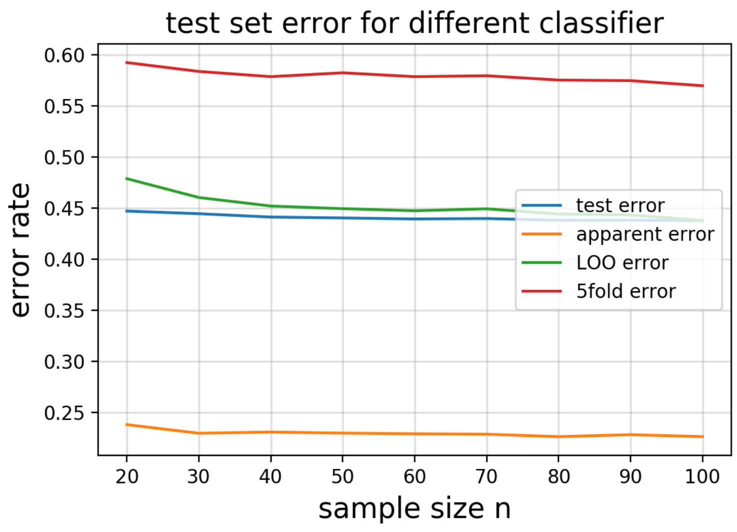


(b) $\sigma = 2$

Figure 2: Error estimation for LDA classifier

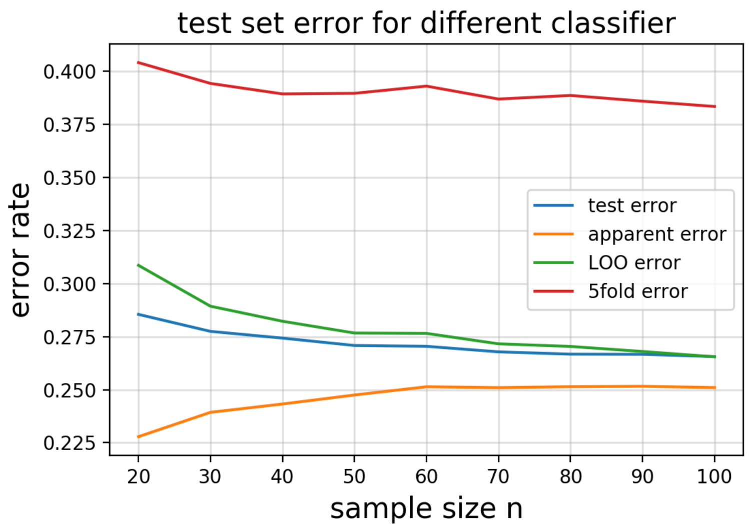


(a) $\sigma = 1$

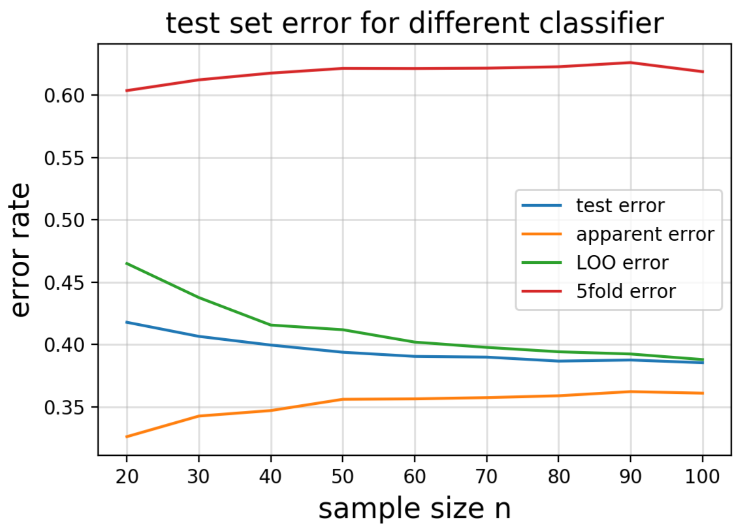


(b) $\sigma = 2$

Figure 3: Error estimation for 3NN classifier



(a) $\sigma = 1$



(b) $\sigma = 2$

Figure 4: Error estimation for SVM classifier

Python Code

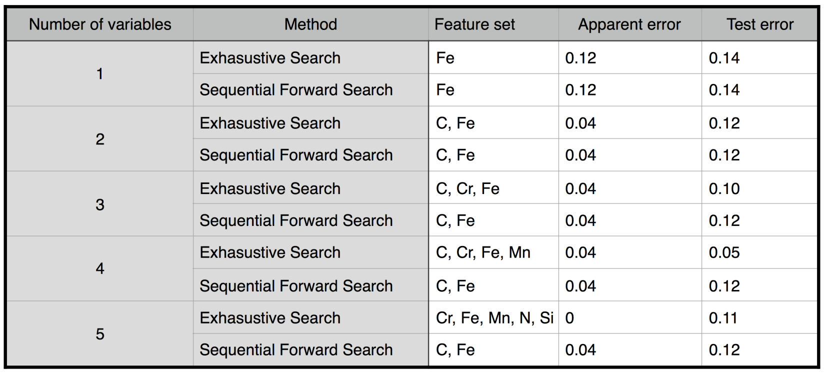
```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import KFold
from scipy.stats import multivariate_normal
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import binom
from sklearn.svm import LinearSVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import LeaveOneOut
from numpy.linalg import inv
from scipy.stats import norm
import math


#get set with size = n and coefficient sigma = sigma
def getSet(n,sigma):
    res = binom.rvs(1,0.5,size = n)
    size0 = sum(res)
    size1 = n - sum(res)
    # reject set whihc has more than 60% '0' or 60 % '1'
    # without this step may occur error when we fit train set with only one label
    while(size0 >= 0.6 * n or size1 >= 0.6 * n):
        res = binom.rvs(1,0.5,size = n)
        size0 = sum(res)
        size1 = n - sum(res)
    mean0 = np.array([0,0])
    mean1 = np.array([1,1])
    cov = np.array([[1,0.2],[0.2,1]]) * sigma * sigma

    sample0 = multivariate_normal.rvs(mean = mean0, cov = cov, size = size0)

    sample1 = multivariate_normal.rvs(mean = mean1, cov = cov, size = size1)
    res = np.zeros(2)

    return sample0,sample1


#compute error of LDA classifier
def errorOfLDA (sample0, sample1,test_sample0,test_sample1,errorToCal):
    n = sample0.shape[0] + sample1.shape[0]
    size1 = sample1.shape[0]
    size0 = sample0.shape[0]
    sample = np.ones((n,2))
    label = np.zeros((n,1))
    sample[0:size1:1] = np.copy(sample1)
    label[0:size1:1] = np.ones((size1,1))
    sample[size1:n:1] = np.copy(sample0)
    label[size1:n:1] = np.zeros((size0,1))

    clf = LinearDiscriminantAnalysis()
    clf.fit(sample, np.ravel(label))
    if (errorToCal == 'all'):
        err_arr = np.ones(2)
    else :
        err_arr = np.ones(1)
    # estimate apparent error
    if (errorToCal == 'all' or errorToCal == 'apparent'):
        ones = 0
        zeros = 0
        for i in range(0,size1):
            ones += clf.predict(sample[i].reshape(1, -1))
        for i in range(size1,n):
            zeros += 1 - clf.predict(sample[i].reshape(1, -1))
        app_error = (n - (ones + zeros)) / n
        err_arr[0] = app_error
    # estimate test set error
    if (errorToCal == 'all' or errorToCal == 'test'):
        ones = 0
        zeros = 0
        test_size_0 = test_sample0.shape[0]
        test_size_1 = test_sample1.shape[0]
        m = test_size_0 + test_size_1
        for i in range(0,test_size_1):
            ones += clf.predict(test_sample1[i].reshape(1, -1))
        for i in range(0,test_size_0):
            zeros += 1 - clf.predict(test_sample0[i].reshape(1, -1))
        test_error = (m - (ones + zeros)) / (test_size_0 + test_size_1)
        err_arr[err_arr.size - 1] = test_error;
    return err_arr


#compute error of 3NN classifier
def errorOf3NN (sample0, sample1,test_sample0,test_sample1,errorToCal):
    n = sample0.shape[0] + sample1.shape[0]
    size1 = sample1.shape[0]
    size0 = sample0.shape[0]
    sample = np.ones((n,2))
    label = np.zeros((n,1))
    sample[0:size1:1] = np.copy(sample1)
    label[0:size1:1] = np.ones((size1,1))
    sample[size1:n:1] = np.copy(sample0)
    label[size1:n:1] = np.zeros((size0,1))

    neigh = KNeighborsClassifier(n_neighbors=3)
    neigh.fit(sample, np.ravel(label))
    if (errorToCal == 'all'):
        err_arr = np.ones(2)
    else :
        err_arr = np.ones(1)
    # estimate apparent error
    if (errorToCal == 'all' or errorToCal == 'apparent'):
        ones = 0
        zeros = 0
        for i in range(0,size1):
            ones += neigh.predict(sample[i].reshape(1, -1))
        for i in range(size1,n):
            zeros += 1 - neigh.predict(sample[i].reshape(1, -1))
        app_error = (n - (ones + zeros)) / n
        err_arr[0] = app_error
    # estimate test set error
    if (errorToCal == 'all' or errorToCal == 'test'):
        ones = 0
        zeros = 0
        test_size_0 = test_sample0.shape[0]
        test_size_1 = test_sample1.shape[0]
        m = test_size_0 + test_size_1
        for i in range(0,test_size_1):
            ones += neigh.predict(test_sample1[i].reshape(1, -1))
        for i in range(0,test_size_0):
            zeros += 1 - neigh.predict(test_sample0[i].reshape(1, -1))
        test_error = (m - (ones + zeros)) / (test_size_0 + test_size_1)
        err_arr[err_arr.size - 1] = test_error;
    return err_arr
```

```python
#compute error of SVM classifier
def errorOfSVM (sample0, sample1,test_sample0,test_sample1,errorToCal):
    n = sample0.shape[0] + sample1.shape[0]
    size1 = sample1.shape[0]
    size0 = sample0.shape[0]
    sample = np.ones((n,2))
    label = np.zeros((n,1))
    sample[0:size1:1] = np.copy(sample1)
    label[0:size1:1] = np.ones((size1,1))
    sample[size1:n:1] = np.copy(sample0)
    label[size1:n:1] = np.zeros((size0,1))

    clf = LinearSVC()
    clf.fit(sample, np.ravel(label))
    if (errorToCal == 'all'):
        err_arr = np.ones(2)
    else :
        err_arr = np.ones(1)
    # estimate apparent error
    if (errorToCal == 'all' or errorToCal == 'apparent'):
        ones = 0
        zeros = 0
        for i in range(0,size1):
            ones += clf.predict(sample[i].reshape(1, -1))
        for i in range(size1,n):
            zeros += 1 - clf.predict(sample[i].reshape(1, -1))
        app_error = (n - (ones + zeros)) / n
        err_arr[0] = app_error
    # estimate test set error
    if (errorToCal == 'all' or errorToCal == 'test'):
        ones = 0
        zeros = 0
        test_size_0 = test_sample0.shape[0]
        test_size_1 = test_sample1.shape[0]
        m = test_size_0 + test_size_1
        for i in range(0,test_size_1):
            ones += clf.predict(test_sample1[i].reshape(1, -1))
        for i in range(0,test_size_0):
            zeros += 1 - clf.predict(test_sample0[i].reshape(1, -1))
        test_error = (m - (ones + zeros)) / (test_size_0 + test_size_1)
        err_arr[err_arr.size - 1] = test_error;
    return err_arr


#compute LOO and 5 fold error for all three classifier
def CVerror(sample0, sample1):
    n = sample0.shape[0] + sample1.shape[0]
    size1 = sample1.shape[0]
    size0 = sample0.shape[0]
    sample = np.ones((n,2))
    label = np.zeros((n,1))
    sample[0:size1:1] = np.copy(sample1)
    label[0:size1:1] = np.ones((size1,1))
    sample[size1:n:1] = np.copy(sample0)
    label[size1:n:1] = np.zeros((size0,1))

    err_arr = np.zeros(6)
    #  computeSVM error
    def LDA_err(sample,label):
        err_arr = np.zeros(2)
        #   compute LOO error
        loo = LeaveOneOut()
        loo_err = 0
        for train_index, test_index in loo.split(sample):
            loo_sample0 = []
            loo_sample1 = []
            for i in train_index:
                if label[i] == 1:
                    loo_sample1.append(sample[i])
                else :
                    loo_sample0.append(sample[i])
            loo_sample0 = np.array(loo_sample0)
            loo_sample1 = np.array(loo_sample1)
            if (label[test_index[0]] == 1 ):
                loo_err += errorOfLDA(loo_sample0,loo_sample1,np.empty(shape=(0,
                    0)),sample[test_index[0]].reshape((1, 2)),'test')
            else:
                loo_err +=
                    errorOfLDA(loo_sample0,loo_sample1,sample[test_index[0]].reshape((1,
                    2)),np.empty(shape=(0, 0)),'test')
        err_arr[0] = loo_err / n
        #   compute 5fold error
        kf = KFold(n_splits=5)
        kf_err = 0
        for train_index, test_index in kf.split(sample):
            kf_sample0 = []
            kf_sample1 = []
            kf_test0 = []
            kf_test1 = []
            for i in train_index:
                if label[i] == 1:
                    kf_sample1.append(sample[i])
                else :
                    kf_sample0.append(sample[i])
            for i in test_index:
                if label[i] == 1:
                    kf_test1.append(sample[i])
                else :
                    kf_test0.append(sample[i])
            kf_sample0 = np.array(kf_sample0)
            kf_sample1 = np.array(kf_sample1)
            kf_test0 = np.array(kf_test0)
            kf_test1 = np.array(kf_test1)
            kf_err += errorOfLDA(kf_sample0,kf_sample1,kf_test0,kf_test1,'test')
        err_arr[1] = kf_err / 5
        return err_arr

    #  computeSVM error
    def kNN_err(sample,label):
        err_arr = np.zeros(2)
        #   compute LOO error
        loo = LeaveOneOut()
        loo_err = 0
        for train_index, test_index in loo.split(sample):
            loo_sample0 = []
            loo_sample1 = []
            for i in train_index:
                if label[i] == 1:
                    loo_sample1.append(sample[i])
                else :
                    loo_sample0.append(sample[i])
            loo_sample0 = np.array(loo_sample0)
            loo_sample1 = np.array(loo_sample1)
            if (label[test_index[0]] == 1 ):
                loo_err += errorOf3NN(loo_sample0,loo_sample1,np.empty(shape=(0,
                    0)),sample[test_index[0]].reshape((1, 2)),'test')
            else:
                loo_err +=
                    errorOf3NN(loo_sample0,loo_sample1,sample[test_index[0]].reshape((1,
                    2)),np.empty(shape=(0, 0)),'test')
        err_arr[0] = loo_err / n
        #   compute 5fold error
        kf = KFold(n_splits=5)
        kf_err = 0
        for train_index, test_index in kf.split(sample):
            kf_sample0 = []
            kf_sample1 = []
            kf_test0 = []
            kf_test1 = []
            for i in train_index:
                if label[i] == 1:
                    kf_sample1.append(sample[i])
                else :
                    kf_sample0.append(sample[i])
            for i in test_index:
                if label[i] == 1:
                    kf_test1.append(sample[i])
                else :
                    kf_test0.append(sample[i])
            kf_sample0 = np.array(kf_sample0)
            kf_sample1 = np.array(kf_sample1)
            kf_test0 = np.array(kf_test0)
            kf_test1 = np.array(kf_test1)
            kf_err += errorOf3NN(kf_sample0,kf_sample1,kf_test0,kf_test1,'test')
        err_arr[1] = kf_err / 5
        return err_arr
#  computeSVM error
    def SVM_err(sample,label):
        err_arr = np.zeros(2)
        #   compute LOO error
        loo = LeaveOneOut()
        loo_err = 0
        for train_index, test_index in loo.split(sample):
            loo_sample0 = []
            loo_sample1 = []
            for i in train_index:
                if label[i] == 1:
                    loo_sample1.append(sample[i])
                else :
                    loo_sample0.append(sample[i])
            loo_sample0 = np.array(loo_sample0)
            loo_sample1 = np.array(loo_sample1)
            if (label[test_index[0]] == 1 ):
                loo_err += errorOfSVM(loo_sample0,loo_sample1,np.empty(shape=(0,
                    0)),sample[test_index[0]].reshape((1, 2)),'test')
            else:
                loo_err +=
                    errorOfSVM(loo_sample0,loo_sample1,sample[test_index[0]].reshape((1,
                    2)),np.empty(shape=(0, 0)),'test')
        err_arr[0] = loo_err / n
        #   compute 5fold error
        kf = KFold(n_splits=5)
        kf_err = 0
        for train_index, test_index in kf.split(sample):
            kf_sample0 = []
            kf_sample1 = []
            kf_test0 = []
            kf_test1 = []
            for i in train_index:
                if label[i] == 1:
                    kf_sample1.append(sample[i])
                else :
                    kf_sample0.append(sample[i])
            for i in test_index:
                if label[i] == 1:
                    kf_test1.append(sample[i])
                else :
                    kf_test0.append(sample[i])
            kf_sample0 = np.array(kf_sample0)
            kf_sample1 = np.array(kf_sample1)
            kf_test0 = np.array(kf_test0)
            kf_test1 = np.array(kf_test1)
            kf_err += errorOfSVM(kf_sample0,kf_sample1,kf_test0,kf_test1,'test')
        err_arr[1] = kf_err / 5
        return err_arr

    LDA_res = LDA_err(sample,label)
    kNN_res = kNN_err(sample,label)
    SVM_res = SVM_err(sample,label)
    err_arr[0] = LDA_res[0]
    err_arr[1] = LDA_res[1]
    err_arr[2] = kNN_res[0]
    err_arr[3] = kNN_res[1]
    err_arr[4] = SVM_res[0]
    err_arr[5] = SVM_res[1]
    return err_arr

# estimate error with order listed below
def estErr(n, M, L,sigma):
    # error for each classifier are ordered in such sequence:test set error,apparent
        error, LOO error and 5 fold error
    errorLDA = np.zeros(4)
    error3NN = np.zeros(4)
    errorSVM = np.zeros(4)
    for i in range(0,M):
        train = getSet(n,sigma)
        test = getSet(L,sigma)
        LDA_12 = errorOfLDA(train[0],train[1],test[0],test[1],'all')
        kNN_12 = errorOf3NN(train[0],train[1],test[0],test[1],'all')
        SVM_12 = errorOfSVM(train[0],train[1],test[0],test[1],'all')
        cv_err = CVerror(train[0],train[1])

        errorLDA[0] += LDA_12[1]
        errorLDA[1] += LDA_12[0]
        errorLDA[2] += cv_err[0]
        errorLDA[3] += cv_err[1]

        error3NN[0] += kNN_12[1]
        error3NN[1] += kNN_12[0]
        error3NN[2] += cv_err[2]
        error3NN[3] += cv_err[3]

        errorSVM[0] += SVM_12[1]
        errorSVM[1] += SVM_12[0]
        errorSVM[2] += cv_err[4]
        errorSVM[3] += cv_err[5]

    return errorLDA / M, error3NN / M, errorSVM / M


# compute error with different n
error_sigma_1_LDA = np.zeros((9,4))
```

```python
error_sigma_1_3NN = np.zeros((9,4))
error_sigma_1_SVM = np.zeros((9,4))
for i in range(0,9):
    est_err = estErr(20 + 10 * i,1,400,1)
    error_sigma_1_LDA[i] = est_err[0]
    error_sigma_1_3NN[i] = est_err[1]
    error_sigma_1_SVM[i] = est_err[2]
print(error_sigma_1_LDA)
print(error_sigma_1_3NN)
print(error_sigma_1_SVM)
# np.savetxt('error_sigma_1_LDA',error_sigma_1_LDA)
# np.savetxt('error_sigma_1_3NN',error_sigma_1_3NN)
# np.savetxt('error_sigma_1_SVM',error_sigma_1_SVM)

error_sigma_2_LDA = np.zeros((9,4))
error_sigma_2_3NN = np.zeros((9,4))
error_sigma_2_SVM = np.zeros((9,4))
for i in range(0,9):
    est_err = estErr(20 + 10 * i,1,400,2)
    error_sigma_2_LDA[i] = est_err[0]
    error_sigma_2_3NN[i] = est_err[1]
    error_sigma_2_SVM[i] = est_err[2]
print(error_sigma_2_LDA)
print(error_sigma_2_3NN)
print(error_sigma_2_SVM)

# np.savetxt('error_sigma_2_LDA.csv',error_sigma_2_LDA)
# np.savetxt('error_sigma_2_3NN.csv',error_sigma_2_3NN)
# np.savetxt('error_sigma_2_SVM.csv',error_sigma_2_SVM)


# plot figure

plt.clf()
n_range = np.linspace(20,100,9)
plt.plot(n_range,error_sigma_2_LDA[:,0],label = 'LDA error')
plt.plot(n_range,error_sigma_2_3NN[:,0],label = '3NN error')
plt.plot(n_range,error_sigma_2_SVM[:,0],label = 'SVM error')
plt.legend(loc='right')
plt.title('test set error for different classifier',fontsize=15)
plt.xlabel('sample size n',fontsize=15)
plt.ylabel('error rate',fontsize=15)
plt.grid(linewidth=1,alpha = 0.4)
# plt.savefig('2_1_sigma_2.png',dpi = 200)
plt.show()


plt.clf()
n_range = np.linspace(20,100,9)
plt.plot(n_range,error_sigma_1_SVM[:,0],label = 'test error')
plt.plot(n_range,error_sigma_1_SVM[:,1],label = 'apparent error')
plt.plot(n_range,error_sigma_1_SVM[:,2],label = 'LOO error')
plt.plot(n_range,error_sigma_1_SVM[:,3],label = '5fold error')
plt.legend(loc='right')
plt.title('test set error for different classifier',fontsize=15)
plt.xlabel('sample size n',fontsize=15)
plt.ylabel('error rate',fontsize=15)
plt.grid(linewidth=1,alpha = 0.4)
# plt.savefig('2_2_sigma_1_SVM.png',dpi = 200)
plt.show()
```

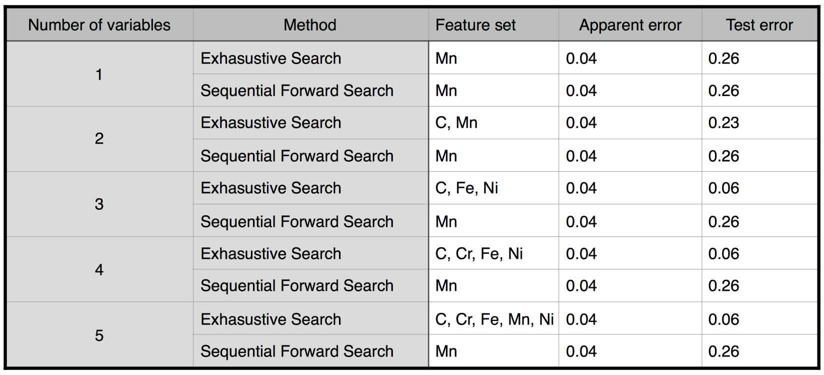
Assignment2

(a)



Number of variables	Method	Feature set	Apparent error	Test error
1	Exhasustive Search	Fe	0.12	0.14
	Sequential Forward Search	Fe	0.12	0.14
2	Exhasustive Search	C, Fe	0.04	0.12
	Sequential Forward Search	C, Fe	0.04	0.12
3	Exhasustive Search	C, Cr, Fe	0.04	0.10
	Sequential Forward Search	C, Fe	0.04	0.12
4	Exhasustive Search	C, Cr, Fe, Mn	0.04	0.05
	Sequential Forward Search	C, Fe	0.04	0.12
5	Exhasustive Search	Cr, Fe, Mn, N, Si	0	0.11
	Sequential Forward Search	C, Fe	0.04	0.12

(a) Features sets selected with error estimate of LDA classifier



Number of variables	Method	Feature set	Apparent error	Test error
1	Exhasustive Search	Mn	0.04	0.26
	Sequential Forward Search	Mn	0.04	0.26
2	Exhasustive Search	C, Mn	0.04	0.23
	Sequential Forward Search	Mn	0.04	0.26
3	Exhasustive Search	C, Fe, Ni	0.04	0.06
	Sequential Forward Search	Mn	0.04	0.26
4	Exhasustive Search	C, Cr, Fe, Ni	0.04	0.06
	Sequential Forward Search	Mn	0.04	0.26
5	Exhasustive Search	C, Cr, Fe, Mn, Ni	0.04	0.06
	Sequential Forward Search	Mn	0.04	0.26

(b) Features sets selected with error estimate of 3NN classifier

Figure 5: Features sets selected with LDA and 3NN as error estimates

For the results found in this project, I found that SFS (sequential forward search) method will soon stop at a feature set of one or two, while exhaustive search method, by definition, will produce growing feature sets. As for filter method used in project 1, we found that the top two predictors(Ni and Fe) form the optimal feature set and if more predictors are added, the test set error will increase. If we neglect that project 1 and project 2 are based on different train set and test set and assume that the result of test set error here is close to the average test set error on randomly selected train set and test set, we could conclude that on general wrapper selection based on LDA classifier has less error rate compared with simple filter selection method while 3NN based wrapper selection has higher error rate. However, for now this is a pure assumption which needs to be validated by more tests

In terms of feature selection and error estimation method, two kinds of classifier- LDA and 3NN are used as error estimators in the wrapper method. Generally speaking, LDA is a better one because it has less average error on test set although the lowest test set error of them is the same. This also agrees with our findings in assignment 1 that in LDA classifier the difference between apparent error and test set error is less than that in 3NN classifier. In fact, 3NN's performance under exhaustive search method is very close to LDA's. However, 3NN performs badly in SFS method. The reason may be that SFS method produces smaller feature sets and 3NN has a dependency on the number of features.

As for feature selection method, generally speaking exhaustive search has less error rate on test set. But there are cases that a smaller feature set could show a lower test set error and thus SFS method is better in test set error. However, we should also be aware that exhaustive search is much more computationally expensive than SFS method so there exist a trade-off between accuracy and efficiency.

If more training points were available, I think the overall performance for both feature selection and error estimation methods would be improved. And because there will be larger feature sets in SFS method, 3NN under SFS method may benefit more. For the feature selection method, it is hard to draw a definite conclusion as it depends on whether more feature will be beneficial, which depends on the intrinsic properties of the test material.

Python code

```python
import matplotlib.pyplot as plt
import pandas as dp
import numpy as np
from sklearn.model_selection import train_test_split
from scipy.stats import ttest_ind
import matplotlib.pyplot as plt
from itertools import chain, combinations
import numpy as np
from scipy.stats import multivariate_normal
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import binom
from sklearn.svm import LinearSVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from numpy.linalg import inv
from scipy.stats import norm
import math

#compute error of LDA
def errorOfLDA (train_set,train_label,test_set,test_label):
    clf = LinearDiscriminantAnalysis()
    clf.fit(train_set, np.ravel(train_label))
    # estimate apparent error
    success = 0
    for i in range(0,test_set.shape[0]):
        if clf.predict(test_set.loc[i].values.reshape(1, -1)).item() == test_label.loc[i]:
            success +=1
    error = 1 - success / test_set.shape[0]
    return error
#compute error of 3NN
def errorOf3NN (train_set,train_label,test_set,test_label):
    neigh = KNeighborsClassifier(n_neighbors=3)
    neigh.fit(train_set, np.ravel(train_label))
    # estimate apparent error
    success = 0
    for i in range(0,test_set.shape[0]):
        if neigh.predict(test_set.loc[i].values.reshape(1, -1)).item() ==
             test_label.loc[i]:
            success +=1
    error = 1 - success / test_set.shape[0]
    return error
# exhaustive search based on given train_set, error_estimator and number of features
def exhaustiveSearch (train_set, est, n_features):
    total_features = train_set.columns.size - 1
    iterlist = []
    for i in range(0,total_features):
        iterlist.append(i)
    subsets = combinations(iterlist,n_features)
    minErr = 1
    optimalSet = []
    for subset in subsets:
#         print (subset)
        currErr =
            est(train_set[list(subset)],train_set[train_set.columns[total_features]],train_set[list(subset)],
        if currErr < minErr:
            minErr = currErr
            optimalSet = subset
        elif currErr == minErr:
            currNum = int(''.join(map(str,subset)))
            minNum = int(''.join(map(str,optimalSet)))
            if (currNum < minNum):
                minErr = currErr
                optimalSet = subset
    return optimalSet,minErr
# SFS based on given train_set, error_estimator and number of features
def SFS (train_set, est, n_features):
    return SFS_helper(train_set,est,n_features,[],1)

def SFS_helper (train_set, est, n_features,curr_features,minErr):
    if len(curr_features) == n_features:
        return curr_features,minErr
    else:
        total_features = train_set.columns.size - 1
        iterlist = []
        for i in range(0,total_features):
            iterlist.append(i)
        iterlist = [x for x in iterlist if x not in curr_features]
        optimalSet = curr_features
        prevMinErr = minErr
        label = train_set.columns[total_features]
        for feature in iterlist:
            curr_features.append(feature)
            currErr =
                 est(train_set[list(curr_features)],train_set[label],train_set[list(curr_features)],train_set[l
#             print(curr_features)
#             print(currErr)
#             print('-----')
            if currErr < minErr:
                minErr = currErr
                optimalSet = list(curr_features)
            elif currErr == minErr:
                currNum = int(''.join(map(str,sorted(curr_features))))
                minNum = int(''.join(map(str,sorted(optimalSet))))
                if (currNum < minNum):
                    minErr = currErr
                    optimalSet = list(curr_features)
            curr_features.remove(curr_features[len(curr_features) - 1])
        if(minErr == prevMinErr) :
            return curr_features,minErr
        else:
            return SFS_helper(train_set,est,n_features,optimalSet,minErr)

#select st - end variables based on train_set, test_set, error_estimator
#and methods(exhaustive search or SFS)
def selectVariable (st, end, train_set, test_set, errEst, methods):
    features = []
    for i in range(st - 1,5):
        for method in methods:
            new = []
            a,b = method(train_set,errEst ,i + 1)
            a = train_set.columns[list(a)]
            c = errEst(train_set[a],train_set['SFE'],test_set[a],test_set['SFE'])
            new.append(sorted(list(a)))
            new.append(b)
            new.append(c)
            features.append(new)
    return features

train_set = dp.read_csv("SFE_Train_Data.csv")
test_set = dp.read_csv("SFE_Test_Data.csv")

#sort by alphabetical order
train_set = train_set[['C','Cr','Fe','Mn','N','Ni','Si','SFE']]
test_set = test_set[['C','Cr','Fe','Mn','N','Ni','Si','SFE']]
```

```
methods = [exhaustiveSearch,SFS]
feature_LDA = selectVariable(1,5,train_set,test_set,errorOfLDA,methods)
feature_3NN = selectVariable(1,5,train_set,test_set,errorOf3NN,methods)

table_LDA = dp.DataFrame(feature_LDA,columns=['feature set','apparent error','test
    error'])
table_LDA.to_csv('lda.csv')

table_3NN = dp.DataFrame(feature_3NN,columns=['feature set','apparent error','test
    error'])
table_3NN.to_csv('3nn.csv')
```