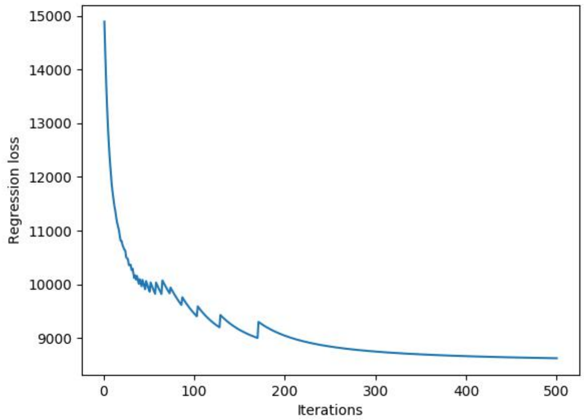
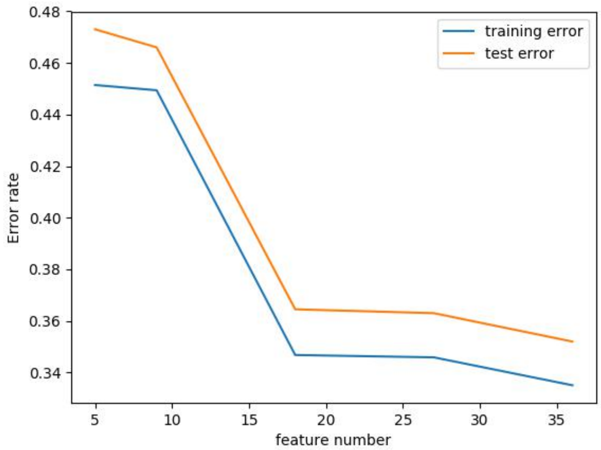
A)

In this assignment, the loss function was chosen as Vapnik loss function. The satimage data set has been analyzed with FSA algorithm, with step size adjusted to be 0.001/N to get the best performance. The training loss vs iteration number has been plotted as below when k=9:



B) The misclassification error vs the number of selected features has been plotted below:

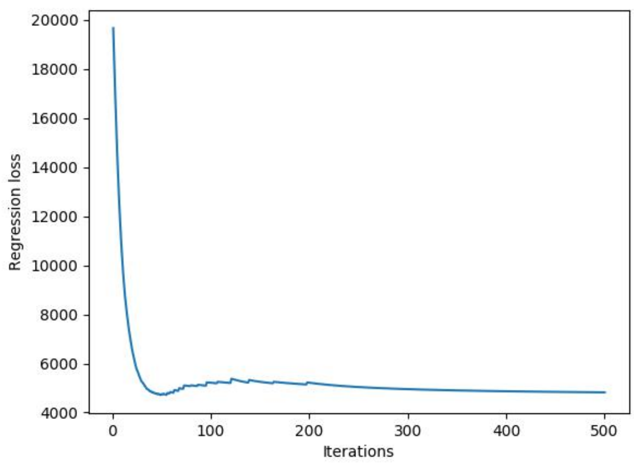


We can see that when more features selected, the overfitting becomes more significant

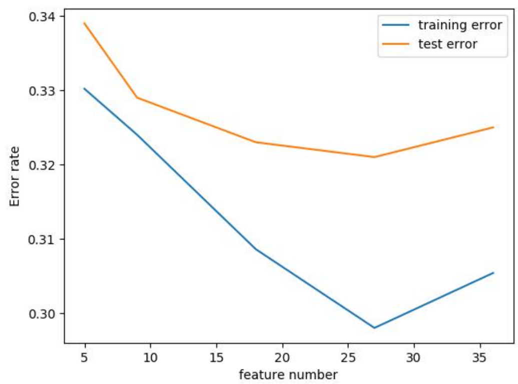
The detailed result has been shown in the table below (together with the result of madelon and arcene dataset):

Parameter number	Train error (satimage)	Test error (satimage)	Train error (covtype)	Test error (covtype)
5	0.4514	0.473	0.3302	0.339
9	0.4494	0.466	0.324	0.329
18	0.3468	0.3645	0.3086	0.323
27	0.3459	0.363	0.298	0.321
36	0.3351	0.352	0.3054	0.325

C)
The algorithm has been used on covtype data set.
The training loss vs iteration number has been plotted as below when k=9:



The misclassification error vs the number of selected features has been plotted below:



The code is as below:

```
import numpy as np
import heapq
import pandas as pd
import math
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from scipy import stats
# reading data from the file
train_data=pd.read_csv('X.dat', sep=' ', header=None).dropna(1).as_matrix()
train_labels=pd.read_csv('Y.dat', sep=' ', header=None).as_matrix()
test_data=pd.read_csv('Xtest.dat', sep=' ', header=None).dropna(1).as_matrix()
test_labels=pd.read_csv('Ytest.dat', sep=' ', header=None).as_matrix()
```

```python
#normalize the data to mean 0 and std 1
def normalize(train, test):
    mean=np.mean(train, axis=0)
    std= np.std(train, axis=0)
    train=(train-mean)/(std+1e-7)
    test=(test-mean)/(std+1e-7)
    return train, test

train_data_norm, test_data_norm =normalize(train_data, test_data)

#take out the data size
N =  train_data_norm.shape[0] #row size
NN = train_data_norm.shape[1] #column size
TN = test_data_norm.shape[0]


#add one extra column 1s at the beginning of the data
train_data = np.hstack((np.ones((N, 1)), train_data_norm))
test_data = np.hstack((np.ones((TN, 1)), test_data_norm))

# train_labels_pro = stats.threshold(train_labels, threshmin = 1)
# test_labels_pro = stats.threshold(test_labels, threshmin = 1)

# train_labels = train_labels_pro
# test_labels = test_labels_pro

def penalty_theta(x,mu, k, i, n):
    temp = np.zeros(x.shape)
    dn=x.shape[1]
    x=x.flatten()
    M=len(x)
    m=k+(M-k)*max(0,(n-2*i)/(2*i*mu+n))
    index=heapq.nlargest(int(m), range(M), np.absolute(x).take)
    for i in index:
        temp[int((i-i%dn)/dn)][i%dn]=x[i]
    return temp

def dl(x):
    if x>1:
        return 0
    else:
        t=x-1
        return 2*t/(1+t*t)

def derivative(w,x,y):
    temp=np.zeros(w.shape)
    l=w.shape[1]
    n=y.shape[0]
    for s in range(l):
        for i in range(n):
            if s==y[i][0]:
                for k in range(l):
                    if k!=s:
temp[:,s]=temp[:,s]+dl(x[i].dot(w[:,s]-w[:,k]))*np.transpose(x[i])
            else:
                temp[:, s] = temp[:, s]
-dl(x[i].dot(w[:,y[i][0]]-w[:,s]))*np.transpose(x[i])
    return temp

def lo(x):
    if x>1:
        return 0
    else:
        return np.log(1+(x-1)*(x-1))

def trainloss(x,y,w):
    l=w.shape[1]
```

```
    n=y.shape[0]
    rt=0
    for i in range(n):
        for k in range(l):
            if k!=y[i][0]:
                rt=rt+lo(x[i].dot(w[:,y[i][0]]-w[:,k]))
    return rt+0.001*np.linalg.norm(x)**2

def iteration_steps(train_data_, train_label_, w, steps, mu, k ,step_size):
    N = train_data_.shape[0]
    steplist=[]
    lostlist=[]
    for i in range(steps):
        w_temp = w -derivative(w, train_data_,
train_label_)*step_size/N-2*0.001*step_size/N*w
        w_temp = penalty_theta(w_temp, mu, k, i, steps)
        w = w_temp
        steplist.append(i+1)
        lostlist.append(trainloss(train_data_, train_label_,w))

    if(k==9):
        plt.plot(steplist, lostlist)
        plt.xlabel('Iterations')
        plt.ylabel('Regression loss')
        plt.legend(loc=1)
        plt.show()

    return w

def linear_regression_predit(w, test_data_):
    p = np.argmax(test_data_.dot(w), axis=1)
    return np.expand_dims(p, axis=1)

# 1, 0.2
step_size=0.001
xlabel=[]
train_err_list = []
test_err_list=[]
iteration=500
k_list=[5, 9, 18, 27,36]
mu=100
L=6

for i in k_list:
    w = np.zeros((NN + 1,L))
    w = iteration_steps(train_data, train_labels, w, iteration, mu, i , step_size)
    xlabel.append(i)
    #predicting
    train_pred=np.asarray(linear_regression_predit(w, train_data))
    test_pred=np.asarray(linear_regression_predit(w, test_data))
    train_error=1-accuracy_score(train_labels, train_pred)
    test_error=1-accuracy_score(test_labels, test_pred)
    print('parameter number: ',i, 'train error: ', train_error, 'test error: ',
test_error)
    #recording
    train_err_list.append(train_error)
    test_err_list.append(test_error)


plt.plot(xlabel, train_err_list,label='training error')
plt.plot(xlabel, test_err_list,label='test error')
plt.xlabel('feature number')
plt.ylabel('Error rate')
plt.legend(loc=1)
plt.show()
```