

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [3]:

```
X = pd.read_csv('X.csv', header = None)
y = pd.read_csv('y.csv', header = None)
```

In [4]:

```
# creat 10 folds
from sklearn.model_selection import KFold
kf = KFold(n_splits =10)
kf.get_n_splits(X)
```

Out[4]:

10

2a naive bayes

In [5]:

```
from scipy.special import gamma
import math
tp, tn, fp, fn =0, 0, 0, 0
average_lam0 = 0
average_lam1 = 0

for train_index, test_index in kf.split(X):
    #print("train:", train_index, "Test", test_index)
    X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
    y_train, y_test = y.iloc[train_index,:], y.iloc[test_index,:]
    n = X_train.shape[0]
    columns = X_train.shape[1]
    pi = y_train[0].sum()/n
    n0 = n-y_train[0].sum()
    n1 = y_train[0].sum()
    sum0, sum1 = 0,0
    for i in range(n):
        if y_train.iloc[i,0] == 0:
            sum0 = sum0 + X_train.iloc[i,:]
        else:
            sum1 = sum1 + X_train.iloc[i,:]
    lamb0 = (1+sum0)/(1+n0)
    lamb1 = (1+sum1)/(1+n1)
    average_lam0 = average_lam0+lamb0
    average_lam1 = average_lam1+lamb1
    # predict y value
    y_pred = np.zeros((y_test.shape[0],))
    prior =[1-pi,pi]
    for i in range(y_test.shape[0]):
        p0 = prior[0]
        p1 = prior[1]
        for j in range(columns):
            p0=p0*lamb0[j]**X_test.iloc[i,j]*math.exp(-lamb0[j])/gamma(X_test.iloc[i,j]+1)
            p1=p1*lamb1[j]**X_test.iloc[i,j]*math.exp(-lamb1[j])/gamma(X_test.iloc[i,j]+1)
        y_pred[i] = int(p1 > p0)
        #print(int(p1 > p0))
        if y_pred[i] ==1 and y_test.iloc[i,0] == 1:
            tp +=1
        elif y_pred[i] ==1 and y_test.iloc[i,0] == 0:
            fp +=1
        elif y_pred[i] ==0 and y_test.iloc[i,0] == 0:
            tn +=1
        else:
```

```

        fn +=1
accuracy = (tp + tn)/4600
print ("accuracy:", accuracy)
print("true positive", tp)
print("true negative", tn)
print("false negative", fn)
print("false positive", fp)

```

C:\Users\Mingfeng\Anaconda3\lib\site-packages\ipykernel_launcher.py:33: RuntimeWarning: overflow encountered in double_scalars
C:\Users\Mingfeng\Anaconda3\lib\site-packages\ipykernel_launcher.py:33: RuntimeWarning: invalid value encountered in double_scalars

```

accuracy: 0.8541304347826087
true positive 1699
true negative 2230
false negative 114
false positive 557

```

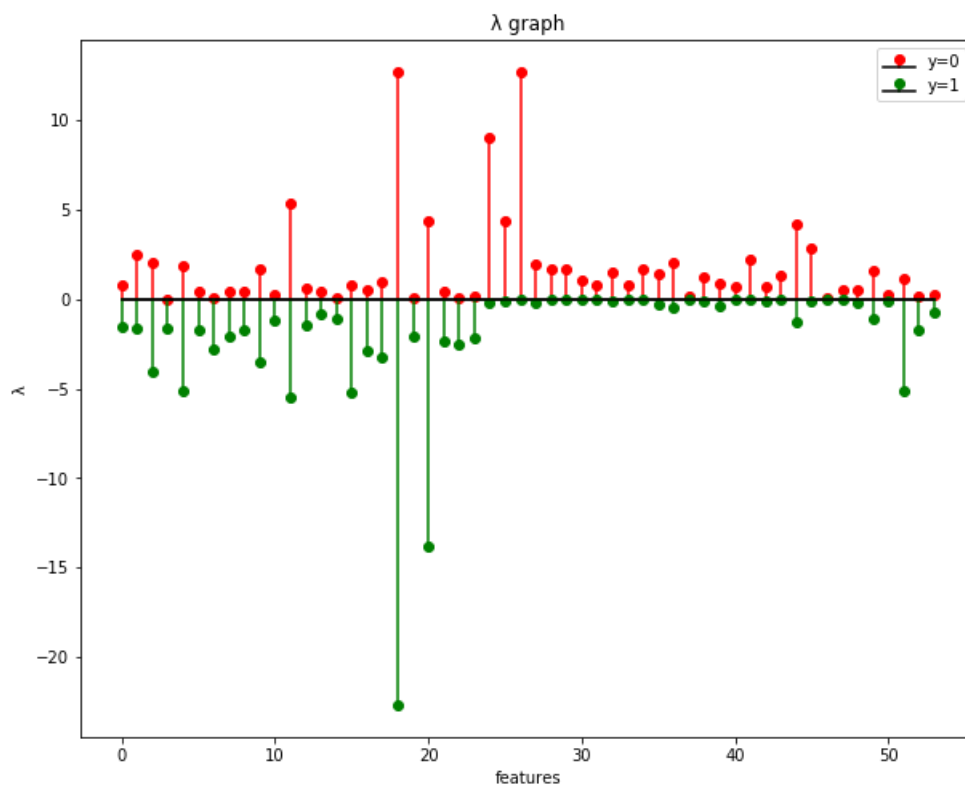
2b stem plot

In [6]:

```

plt.figure(figsize=(10, 8))
average_lam0 = average_lam0/10
average_lam1 = average_lam1/10
plt.stem(np.arange(54), average_lam0, 'r', markerfmt='ro', basefmt='black', label='y=0' )
plt.stem(np.arange(54), -average_lam1, 'g', markerfmt='go', basefmt='black', label='y=1')
plt.legend()
plt.title('\lambda graph')
plt.xlabel('features')
plt.ylabel('\lambda')
plt.show()
#plt.savefig('2b.png')

```



2c KNN

In [7]:

```

def KNN (X_train, X_test, y_train, y_test):
    accuracy = []

```

```

error = []
X_train=X_train.values
X_test=X_test.values
y_train = np.squeeze(np.asarray(y_train))
y_test = np.squeeze(np.asarray(y_test))
y_predict = np.zeros((y_test.shape[0],20))
for i in range(X_test.shape[0]):
    dist = np.sum(np.absolute(X_train-X_test[i,:]),axis=1)
    for k in range(1,21):
        y_predict[i,k-1] = np.argmax(np.bincount(y_train[np.argsort(dist, k-1)[:k]]))
error = np.sum(np.absolute(y_predict-y_test.reshape(-1,1)),axis = 0)
accuracy = 1-error/460
return accuracy

```

In [8]:

```

%%time
ave_accuracy =0
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
    y_train, y_test = y.iloc[train_index,:], y.iloc[test_index,:]
    accuracy_list = KNN(X_train, X_test, y_train, y_test)
    ave_accuracy = ave_accuracy+np.array(accuracy_list)
ave_accuracy =ave_accuracy/10

```

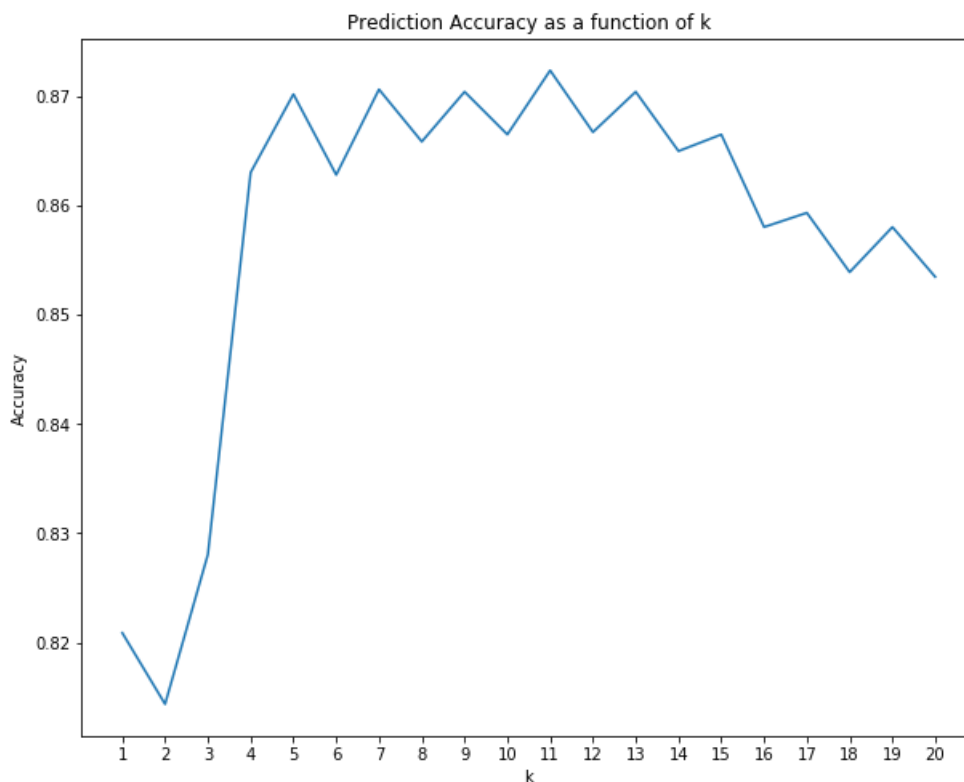
Wall time: 17.9 s

In [9]:

```

plt.figure(figsize=(10, 8))
plt.plot(np.arange(1,21), ave_accuracy)
plt.xticks(range(1,21))
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title('Prediction Accuracy as a function of k')
plt.savefig('2c.png')

```



2d Logistics Regression steepest ascent

In [10]:

```
ite = 1000
```

```
step = 0.01/4600
```

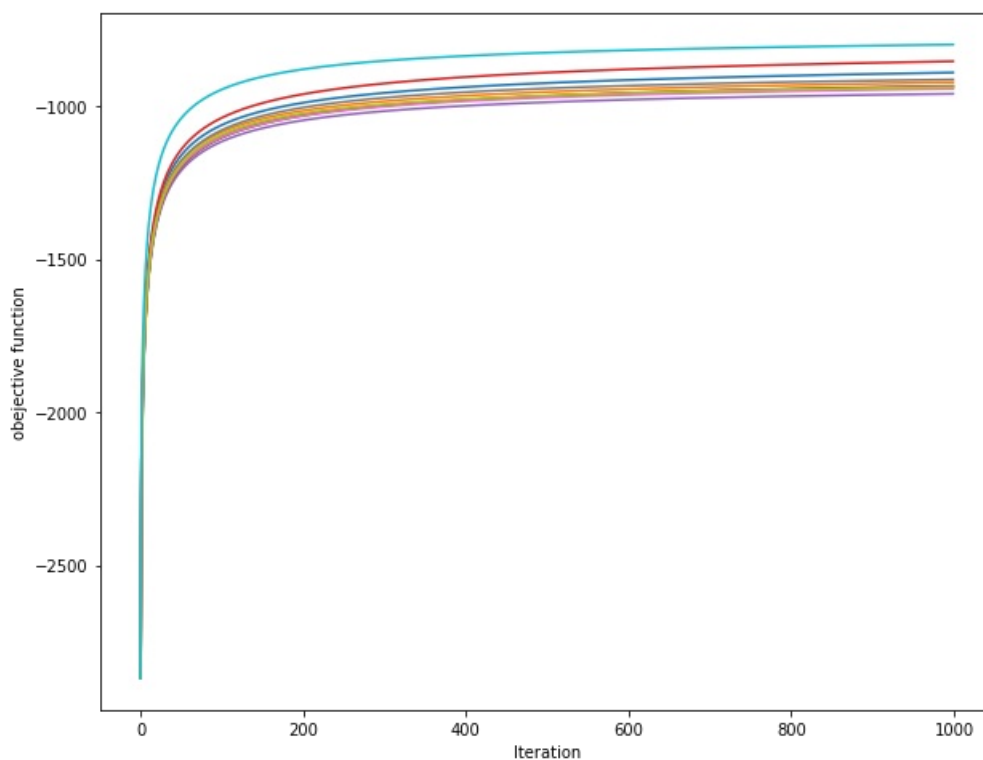
```
In [11]:
```

```
def logit_r(X_train, X_test, y_train, y_test):
    y_train2 = y_train.copy()
    y_test2 = y_test.copy()
    y_train2[y_train2 == 0] = -1
    y_test2[y_test2 == 0] = -1
    X_train2 = np.full((X_train.shape[0], X_train.shape[1] + 1), 1)
    X_train2[:, :-1] = X_train.values
    X_test2 = np.full((X_test.shape[0], X_test.shape[1] + 1), 1)
    X_test2[:, :-1] = X_test.values
    y_train2 = np.squeeze(np.asarray(y_train2))
    y_test2 = np.squeeze(np.asarray(y_test2))
    w = np.zeros((X_train2.shape[1],))
    delta = np.zeros((X_train2.shape[1],))
    #w = np.zeros((1000,))
    L = np.zeros((1000,))
    for t in range(ite):
        log_odd = np.exp(X_train2 * y_train2.reshape(-1, 1) @ w)
        obs_p = log_odd / (1 + log_odd)
        delta = np.sum(X_train2 * y_train2.reshape(-1, 1) * (1 - obs_p).reshape(-1, 1), axis=0)
        w = w + step * delta
        L[t] = np.sum(np.log(obs_p))
    return w, L, X_test2, y_test2
```

```
In [12]:
```

```
%%time
plt.figure(figsize=(10, 8))
plt.xlabel('Iteration')
plt.ylabel('objective function')
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index, :], X.iloc[test_index, :]
    y_train, y_test = y.iloc[train_index, :], y.iloc[test_index, :]
    w, L, X_test2, y_test2 = logit_r(X_train, X_test, y_train, y_test)
    plt.plot(np.arange(1000), L)
plt.savefig('2d.png')
```

Wall time: 1min 24s



```
In [323]:
```

```
plt.savefig('2d.png')
```

<Figure size 432x288 with 0 Axes>

2e Newton's method

In [19]:

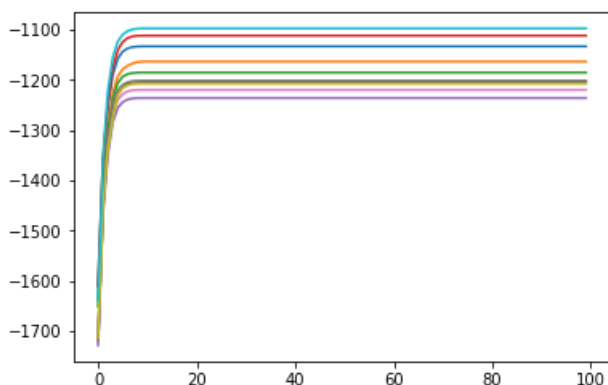
```
def newton_m(X_train, X_test, y_train, y_test):
    y_train2 = y_train.copy()
    y_test2 = y_test.copy()
    y_train2[y_train2 == 0] = -1
    y_test2[y_test2 == 0] = -1
    X_train2 = np.full((X_train.shape[0], X_train.shape[1]+1), 1)
    X_train2[:, :-1] = X_train.values
    X_test2 = np.full((X_test.shape[0], X_test.shape[1]+1), 1)
    X_test2[:, :-1] = X_test.values
    y_train2 = np.squeeze(np.asarray(y_train2))
    y_test2 = np.squeeze(np.asarray(y_test2))

    w = np.zeros((X_train2.shape[1],))
    sigmoid = np.exp((y_train2 * X_train2.T).T @ w) / (1 + np.exp((y_train2 * X_train2.T).T @ w))
    l = np.sum(np.log(sigmoid))
    L = np.zeros((100,))
    for t in range(100):
        gradient = np.sum((1-sigmoid).reshape(-1,1) * (y_train2 * X_train2.T).T, axis=0)
        diagonal = np.diag(sigmoid * (1-sigmoid).reshape(-1,1))
        hessian = - X_train2.T * diagonal @ X_train2 - (10**(-2)) * np.identity(55)
        w = w - (gradient.T @ np.linalg.inv(hessian)).T
        sigmoid = np.exp((y_train2 * X_train2.T).T @ w) / (1 + np.exp((y_train2 * X_train2.T).T @ w))
        l += - (gradient.T @ np.linalg.inv(hessian)) @ gradient + 0.5 * (gradient.T @ np.linalg.inv(
            hessian)) @ hessian @ (gradient.T @ np.linalg.inv(hessian)).T
        L[t] = l
    return w, L
```

In [20]:

```
%%time
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
    y_train, y_test = y.iloc[train_index,:], y.iloc[test_index,:]
    w, L = newton_m(X_train, X_test, y_train, y_test)
    plt.plot(np.arange(100), L)
```

Wall time: 2min 22s



2f

In [29]:

```
%%time
tp, tn, fp, fn = 0, 0, 0, 0
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index,:], X.iloc[test_index,:]
```

```

y_train, y_test = y.iloc[train_index,:], y.iloc[test_index,:]
w,L, X_test2, y_test2= logit_r(X_train, X_test, y_train, y_test)
y_pred = w @ X_test2.T
y_pred[np.where(y_pred >=0)] = 1
y_pred[np.where(y_pred < 0)] = 0
fp = fp + np.sum((y_pred - y_test2) == 2, axis=0)
fn = fn + np.sum((y_pred - y_test2) == -1, axis=0)
tp = tp + np.sum((y_pred - y_test2) == 0, axis=0)
tn = tn + np.sum((y_pred - y_test2) == 1, axis=0)
accuracy = (tp + tn) / 4600

```

Wall time: 1min 47s

In [30]:

```

df = pd.DataFrame([[tn,fp],[fn,tp]], index=['Negative Class','Positive Class'], columns=['Predicted Negative','Predicted Positive'])
print(df)
print(accuracy)

```

	Predicted Negative	Predicted Positive
Negative Class	2501	286
Positive Class	222	1591

0.8895652173913043