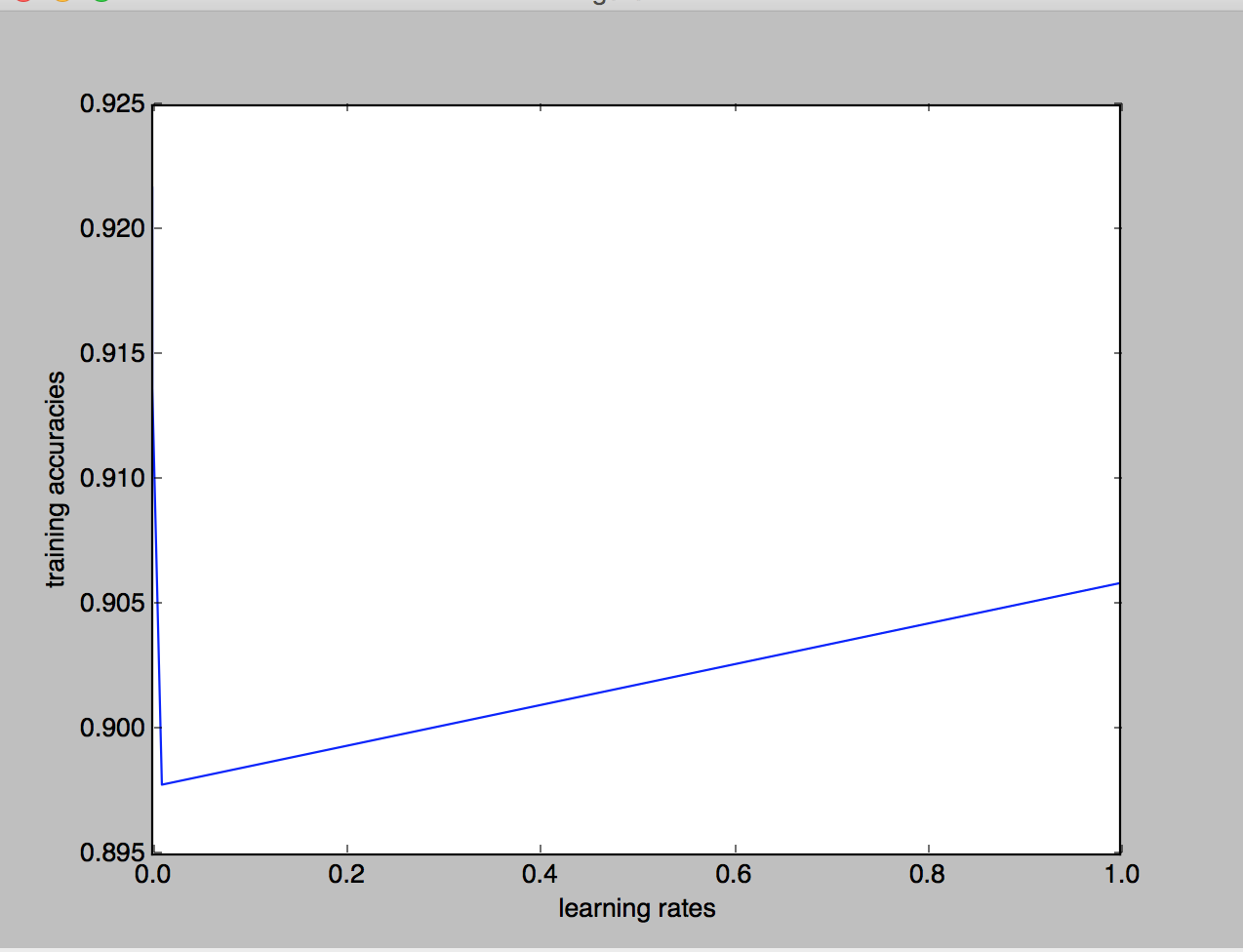
Problem 2

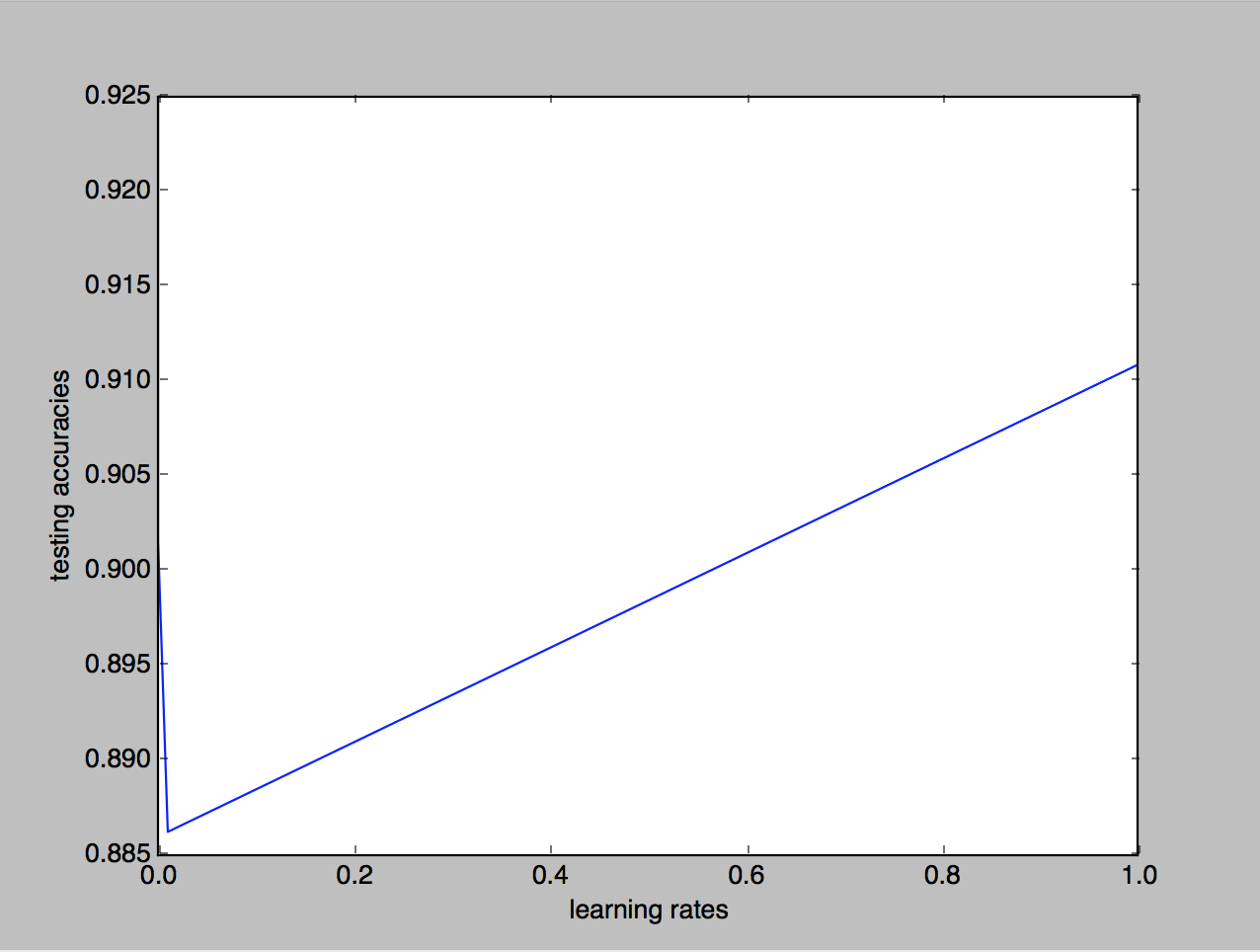
Part a)

Graphs:

Gradient Descent Training:



Gradient descent testing:



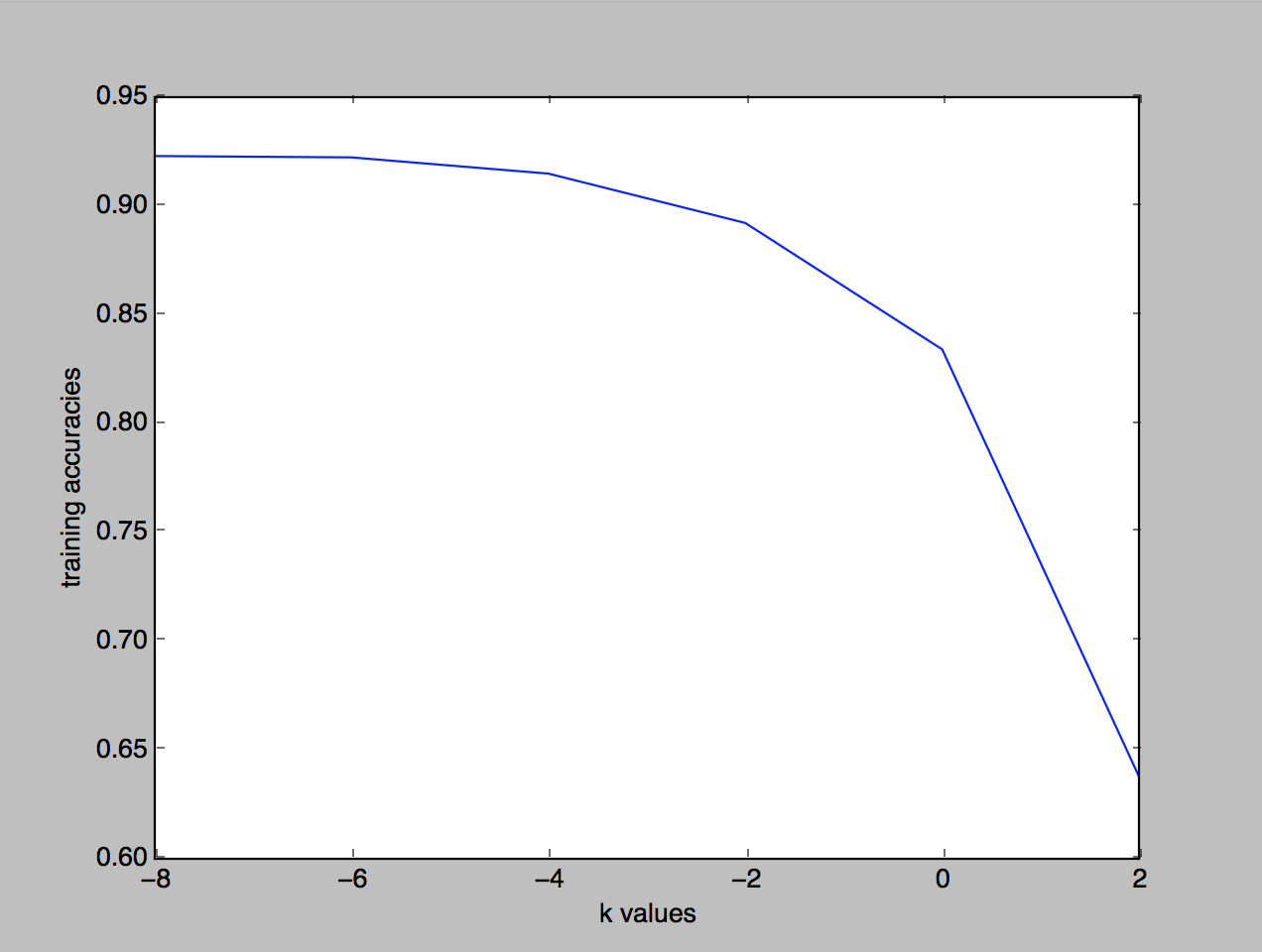
Choosing different learning rates:

When you choose to small of a learning rate, the time to converge is longer the time to converge is longer. This is because, you may get stuck in some local extrema in stuck in some local extrema instead of finding the global extrema.

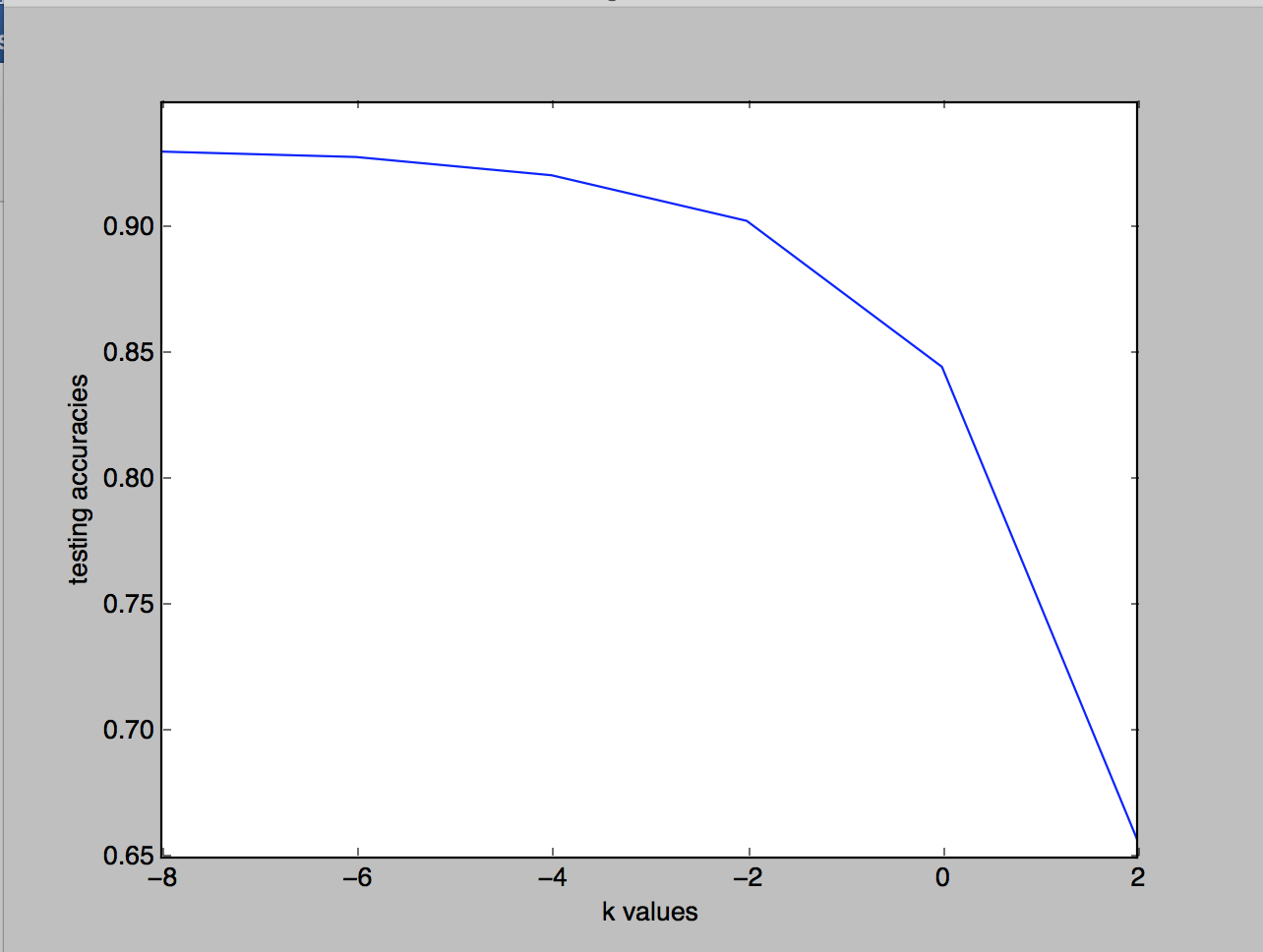
If the learning rate is two high, conversely, you may skip extrema. This would cause the function to not settle in the extrema and oscillate in locals.

B) lambda manipulation

Training:



testing:



It seems as your k values increase, the accuracy tends to decrease significantly. However, the original k value had a significantly higher accuracy then the non-regularized graphs. This disappears as k increases.

Source Code:

I did my program in python because the professor said it was allowed. I’m using the sci-kit learn module for calculations.

Logreg.py:

"""

Octave and julia doesn't work on my mac for some reason

so I did the whole program in python. I used scikit learn

for most all of the bayes and score calculations

"""

import pandas as pd

import numpy as np

import math

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import SGDClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn import metrics

import matplotlib.pyplot as plt

train=pd.read\_csv('/Users/denisdoci/Desktop/spambasetrain.csv')

train\_y=train.iloc[:,-1]

train\_x=train.iloc[:,:len(train.columns)-1]

test=pd.read\_csv('/Users/denisdoci/Desktop/spambasetest.csv')

test\_y=test.iloc[:,-1]

test\_x=test.iloc[:,:len(test.columns)-1]

learning\_rates=[1, .01, 0.0001, 0.000001]

trainingAccuracies = []

testingAccuracies = []

kvalues = [-8,-6,-4,-2,0,2]

kTestingAccuracies = []

kTrainingAccuracies = []

for learning\_rate in learning\_rates:

clf\_LR = SGDClassifier(learning\_rate='optimal', eta0=learning\_rate, penalty = 'none')

clf\_LR.fit(train\_x, train\_y)

score = clf\_LR.score(train\_x, train\_y)

trainingAccuracies.append(score)

clf\_LRT = SGDClassifier(learning\_rate='optimal', eta0=learning\_rate, penalty = 'none')

clf\_LRT.fit(test\_x, test\_y)

score = clf\_LRT.score(test\_x, test\_y)

testingAccuracies.append(score)

for k in kvalues:

clf\_LR = SGDClassifier(learning\_rate='optimal', penalty='l2', alpha=math.pow(2,k))

clf\_LR.fit(train\_x, train\_y)

score = clf\_LR.score(train\_x, train\_y)

kTrainingAccuracies.append(score)

clf\_LRT = SGDClassifier(learning\_rate='optimal', penalty = 'l2', alpha=math.pow(2,k))

clf\_LRT.fit(test\_x, test\_y)

score = clf\_LRT.score(test\_x, test\_y)

kTestingAccuracies.append(score)

print(trainingAccuracies)

print(testingAccuracies)

plt.plot(learning\_rates, trainingAccuracies)

plt.ylabel('training accuracies')

plt.xlabel('learning rates')

plt.show()

plt.plot(learning\_rates, testingAccuracies)

plt.ylabel('testing accuracies')

plt.xlabel('learning rates')

plt.show()

plt.plot(kvalues, kTrainingAccuracies)

plt.ylabel('training accuracies')

plt.xlabel('k values')

plt.show()

plt.plot(kvalues, kTestingAccuracies)

plt.ylabel('testing accuracies')

plt.xlabel('k values')

plt.show()