**Problem 1:**

p(D|theta) =

l(theta) = ln p(D|theta)

gradient\_theta( l(theta) ) =

=

0 = n / theta\_hat -

sum(xi) = n / theta\_hat

theta\_hat = n / sum(xi)

theta\_hat = 1 / (sum(xi) \* 1/n)

Averages for 10 runs each (code attached):

Max Likelihood Estimation

Average: 75.78%

Std. Dev: 2.302

Naive-Bayes

Average: 73.65%

Std. Dev: 2.409

k-Nearest Neighbor k = 1

Average: 64.92%

Std. Dev: 1.680

k-Nearest Neighbor k = 5

Average: 65.49%

Std. Dev: 1.100

k-Nearest Neighbor k = 11

Average: 64.97%

Std. Dev: 2.832

Parzen Window edge length = 20

Average: 65.03%

Std. Dev: 1.610

**Problem 2 (Maximum Likelihood Estimation):**

import csv

import math

import numpy as np

runs = 0

accuracy = []

while (runs < 10):

raw\_data = np.genfromtxt("pima-indians-diabetes.csv", delimiter = ",", usecols = (1, 2, 3, 8))

np.random.shuffle(raw\_data) #randomize data

train\_data = raw\_data[0:len(raw\_data)/2] #assign 1st half of data to training data

test\_data = raw\_data[len(raw\_data)/2:] #assign 2nd half of data to test data

correct = 0

wrong = 0

train\_data\_filter1 = train\_data[:, 3] = train\_data[:, 3] == 1 #setting filter for training data when column 9 = 1

mean1 = train\_data[train\_data\_filter1].mean(0) #calculate mean of each attribute

mean1 = np.delete(mean1, 3, 0) #remove the 4th column from mean vector

var1 = np.cov(train\_data[train\_data\_filter1].transpose()) #calculate covariance matric

var1 = np.delete(var1, 3, 1) #removes the 4th column from the covariance matrix

var1 = np.delete(var1, 3, 0) #removes the 4th row from the covariance matrix

train\_data\_filter0 = train\_data[:, 3] = train\_data[:, 3] == 0

mean0 = train\_data[train\_data\_filter0].mean(0)

mean0 = np.delete(mean0, 3, 0)

var0 = np.cov(train\_data[train\_data\_filter0].transpose())

var0 = np.delete(var0, 3, 1)

var0 = np.delete(var0, 3, 0)

priortmp1 = train\_data[train\_data\_filter1].shape[0] #gives the count where training data column 4 equals 1

priortmp0 = train\_data[train\_data\_filter0].shape[0]

prior0 = float(priortmp0) / (priortmp0 + priortmp1) #calculates prior probability of trainging data where column 4 equals 0

prior1 = float(priortmp1) / (priortmp0 + priortmp1)

constant1 = 1 / math.sqrt(np.linalg.det(var1)) #calculates 1/sqrt(covariance)

constant0 = 1 / math.sqrt(np.linalg.det(var0))

inv\_var0 = np.linalg.inv(var0) #inverse of covariance matrix

inv\_var1 = np.linalg.inv(var1)

for i in test\_data:

j = np.delete(i, 3, 0)

diff0 = j - mean0

lklhood0 = constant0 \* np.exp(-np.dot(np.dot(diff0, inv\_var0), diff0) / 2)

diff1 = j - mean1

lklhood1 = constant1 \* np.exp(-np.dot(np.dot(diff1, inv\_var1), diff1) / 2)

post0 = prior0 \* lklhood0

post1 = prior1 \* lklhood1

if(post0 > post1 and i[3] == 0):

correct += 1

elif(post1 < post0 and i[3] == 1):

correct += 1

else:

wrong += 1

accuracy.append(100 \* float(correct) / (correct + wrong))

runs += 1

print "Average Accuracy: " + str(np.average(accuracy))

print "Standard Deviation of Accuracy: " + str(np.std(accuracy))

**Problem 2 (Naive Bayes):**

import csv

import math

import numpy as np

runs = 0

accuracy = []

while(runs < 10):

raw\_data = np.genfromtxt("pima-indians-diabetes.csv", delimiter = ",", usecols = (1, 2, 3, 8))

np.random.shuffle(raw\_data) #randomize raw data

train\_data = raw\_data[0:len(raw\_data)/2] #separate 1st half to training data

test\_data = raw\_data[len(raw\_data)/2:] #separate 2nd half to test data

correct = 0

wrong = 0

train\_data\_filter1 = train\_data[:, 3] = train\_data[:, 3] == 1 #setup filter where training data column 4 = 1

mean1 = train\_data[train\_data\_filter1].mean(0) #calculate mean for each attribute in training data

mean1 = np.delete(mean1, 3, 0) #delete the 4th column from mean vector

var1 = np.cov(train\_data[train\_data\_filter1].transpose()) #calculate covariance matrix on training data

var1 = np.delete(var1, 3, 1) #delete column 4 from covariance matrix

var1 = np.delete(var1, 3, 0) #delete row 4 from covariance matrix

train\_data\_filter0 = train\_data[:, 3] = train\_data[:, 3] == 0 #setup filter where training data column 4 = 0

mean0 = train\_data[train\_data\_filter0].mean(0)

mean0 = np.delete(mean0, 3, 0)

var0 = np.cov(train\_data[train\_data\_filter0].transpose())

var0 = np.delete(var0, 3, 1)

var0 = np.delete(var0, 3, 0)

priortmp1 = train\_data[train\_data\_filter1].shape[0] #number of training data records where column 4 = 1

priortmp0 = train\_data[train\_data\_filter0].shape[0] #number of training data records where column 4 = 0

#calculate priors

prior0 = float(priortmp0) / (priortmp0 + priortmp1)

prior1 = float(priortmp1) / (priortmp0 + priortmp1)

for i in test\_data:

#calculate the independent likelihoods

lklhood00 = np.exp(-(i[0] - mean0[0]) \* (i[0] - mean0[0]) / (2 \* var0[0,0])) / math.sqrt(var0[0,0])

lklhood01 = np.exp(-(i[1] - mean0[1]) \* (i[1] - mean0[1]) / (2 \* var0[1,1])) / math.sqrt(var0[1,1])

lklhood02 = np.exp(-(i[2] - mean0[2]) \* (i[2] - mean0[2]) / (2 \* var0[2,2])) / math.sqrt(var0[2,2])

lklhood10 = np.exp(-(i[0] - mean1[0]) \* (i[0] - mean1[0]) / (2 \* var1[0,0])) / math.sqrt(var1[0,0])

lklhood11 = np.exp(-(i[1] - mean1[1]) \* (i[1] - mean1[1]) / (2 \* var1[1,1])) / math.sqrt(var1[1,1])

lklhood12 = np.exp(-(i[2] - mean1[2]) \* (i[2] - mean1[2]) / (2 \* var1[2,2])) / math.sqrt(var1[2,2])

#calculate the posteriors

post0 = prior0 \* lklhood00 \* lklhood01 \* lklhood02

post1 = prior1 \* lklhood10 \* lklhood11 \* lklhood12

#make decisions based on the posterior and count correct vs wrong on test data

if(post0 > post1 and i[3] == 0):

correct += 1

elif(post1 < post0 and i[3] == 1):

correct += 1

else:

wrong += 1

runs += 1

accuracy.append(100 \* float(correct) / (correct + wrong))

print "Accuracy Average: " + str(np.average(accuracy))

print "Accuracy Standard Deviation: " + str(np.std(accuracy))

**Problem 3 (k-Nearest Neighbor):**

import csv

import math

import numpy as np

from scipy import spatial

runs = 0

accuracy = []

while(runs < 10):

raw\_data = np.genfromtxt("pima-indians-diabetes.csv", delimiter = ",", usecols = (1, 2, 3, 8))

np.random.shuffle(raw\_data) #shuffle data

num\_samples = 1 #value of k

train\_data = raw\_data[0:len(raw\_data)/2] #separate 1st half of raw data into training data

test\_data = raw\_data[len(raw\_data)/2:] #separate 2nd half of raw data into test data

correct = 0

wrong = 0

post0 = 0

post1 = 0

train = np.delete(train\_data, 3, 1) #assign train to train\_data without column 4

tree = spatial.KDTree(train) #create kd tree with training data

for i in test\_data:

distance, closest = tree.query(np.delete(i, 3, 0), k = num\_samples) #find k nearest neighbor(s) and assign it/them to closest

j = 0

#closest will be a list if k is greater than 1 and integer if equal to 1, so this conditional separates them

if(num\_samples > 1):

while(j < num\_samples): #this checks each of the closest neighbor

if(train\_data[closest[j], 3] == 1):

post1 += 1

else:

post0 += 1

j += 1

else:

if(train\_data[closest, 3] == 1):

post1 += 1

else:

post0 += 1

if(post1 > post0 and i[3] == 1):

correct += 1

elif(post0 > post1 and i[3] == 0):

correct += 1

else:

wrong += 1

runs += 1

accuracy.append(100 \* float(correct) / (correct + wrong))

print "Accuracy Average: " + str(np.average(accuracy))

print "Accuracy Standard Deviation: " + str(np.std(accuracy))

**Problem 4 (Parzen Window):**

import csv

import math

import numpy as np

from scipy import spatial

runs = 0

accuracy = []

while(runs < 10):

raw\_data = np.genfromtxt("pima-indians-diabetes.csv", delimiter = ",", usecols = (1, 2, 3, 8))

np.random.shuffle(raw\_data) #randomizes data

window\_size = 20 #sets radius of hypercube

train\_data = raw\_data[0:len(raw\_data)/2] #separates the 1st half of data to training data

test\_data = raw\_data[len(raw\_data)/2:] #separates the 2nd half of data to testing data

correct = 0

wrong = 0

post0 = 0

post1 = 0

train = np.delete(train\_data, 3, 1) #assign training data without column 4 to train

tree = spatial.KDTree(train) #assign training data to kd tree

for i in test\_data:

closest = tree.query\_ball\_point(np.delete(i, 3, 0), window\_size) #returns the indexes for neighbors inside of window

j = 0

while(j < len(closest)):

if(train\_data[closest[j], 3] == 1):

post1 += 1

else:

post0 += 1

j += 1

if(post1 > post0 and i[3] == 1):

correct += 1

elif(post0 > post1 and i[3] == 0):

correct += 1

else:

wrong += 1

runs += 1

accuracy.append(100 \* float(correct) / (correct + wrong))

print "Accuracy Average: " + str(np.average(accuracy))

print "Accuracy Standard Deviation: " + str(np.std(accuracy))