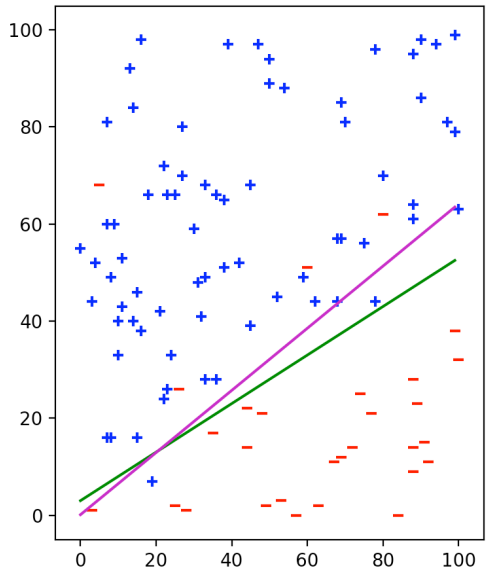
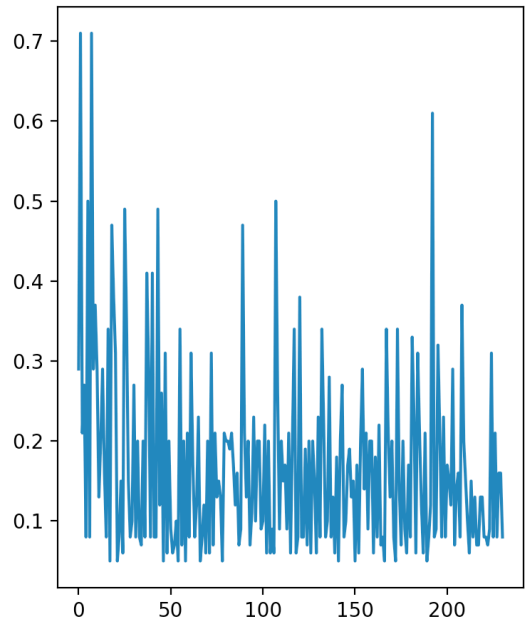
Samantha Rothman

Problem 1- Linear Regression

1) 5% noise
   b) When we run the perceptron algorithm on a random sample of 100 linearly separable two-dimensional points with 5% noise, the graph looks like the one below. The perceptron is in magenta and the target is in green.



When we plot E_in (the number of misclassified points divided by the total number of points) as a function of the 231 iterations that the algorithm used, we can see it decreases but does not go exactly to zero. This, along with the perceptron line not being the most accurate, is because of the noise.

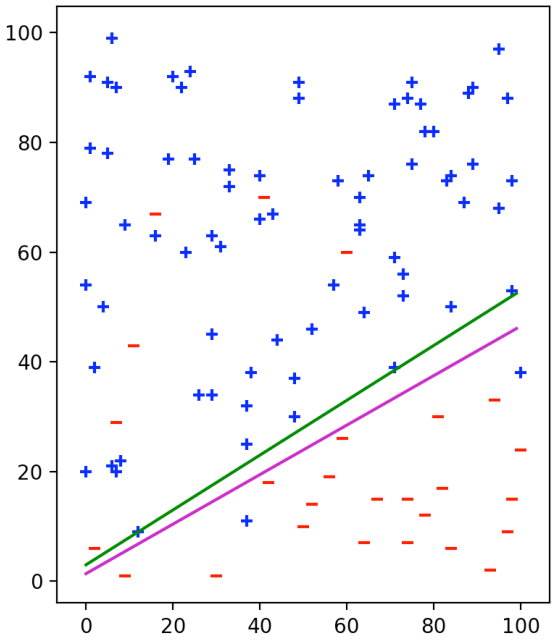


   c) The in-sample error for the linear regression is 0.01591060368873454.

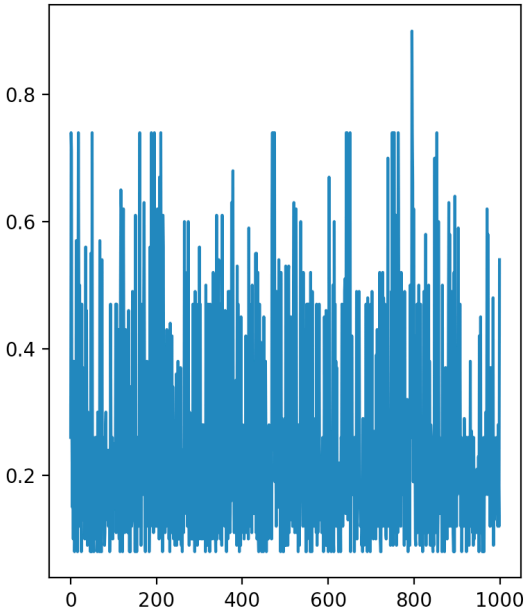
Samantha Rothman

2) 10% noise

b) When we flip 10% of the labels in the sample and run the perceptron, we get the graph below. The perceptron is in magenta and the target is in green.



This time, the perceptron algorithm uses all 1000 iterations because of more noise, so the E_in as a function of iterations looks like this:

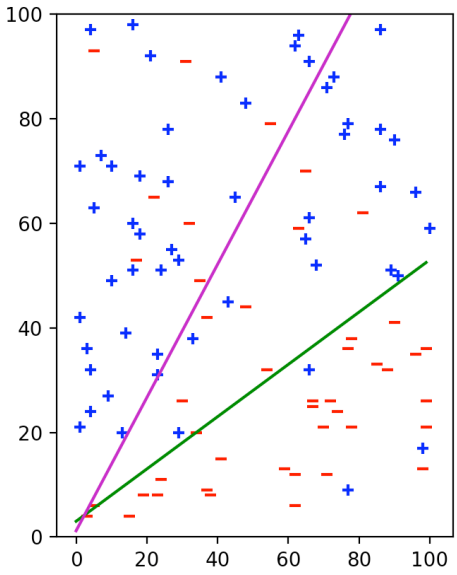


c) The E_in for the linear regression is 0.04550806395708674.

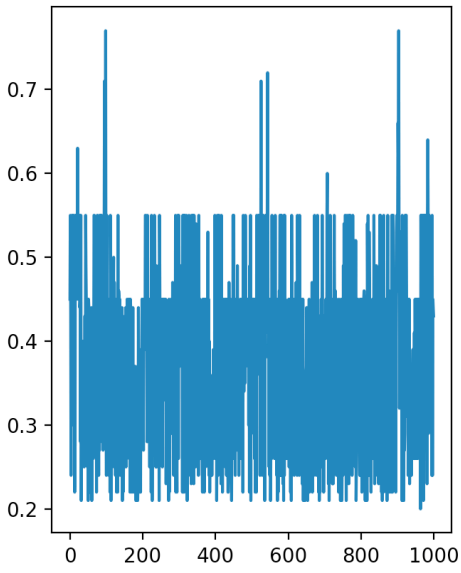
Samantha Rothman

3) 20% noise

b) When we flip 20% of the labels in the sample and run the perceptron, we get the graph below. The perceptron is in magenta and the target is in green. The perceptron definitely does not do as good of a job at separating the data.



Once again, the algorithm uses all 1000 iterations, giving us the E_in graph below because of the noise. The E_in never even goes below 0.2



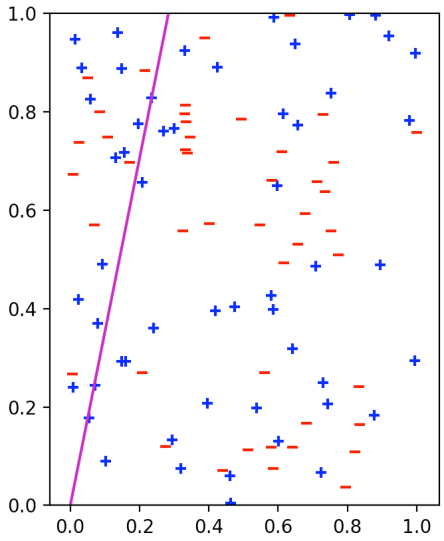
d) Using the linear regression, the E_in is 0.06318663834265603.

Samantha Rothman

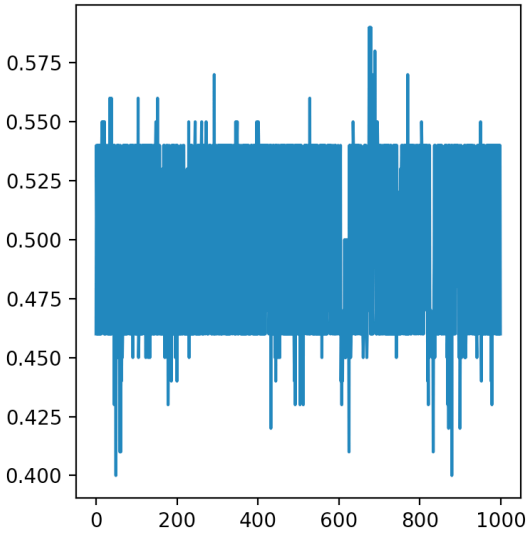
Problem 2- Feature Transform

a) After creating 100 random two-dimensional data points and mapping it to five-dimensional data, I labeled the data using the hyperplane, h(z) = 1 + 3*x + 4*y + 4*(x**2) + 2*(y**2). The two-dimensional points can be seen in part b and are not linearly separable.

b) Since the points are not linearly separable, the perceptron algorithm does not do a good job at dividing the data as can be seen below.



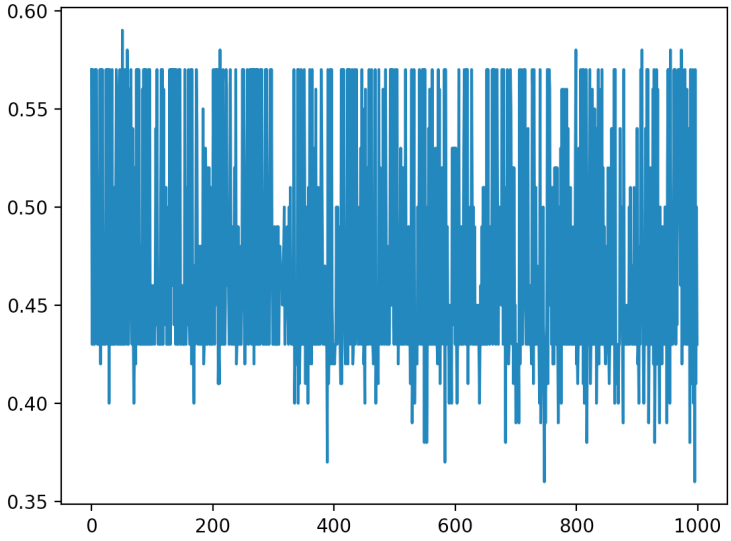
The algorithm uses all 1000 iterations, giving us the E_in graph below. The in-sample error always stays about half because of the fact that the data is not linearly separable.



c) When we run the linear regression on the two-dimensional data, we get that the error is 0.029704998305402436.

Samantha Rothman

d) The perceptron algorithm on the five-dimensional data Z uses all 1000 iterations. The E_in is graphed below as a function of those iterations. It is clearly going down, and with more iterations, it should eventually go to zero.



Samantha Rothman

Code for perceptron algorithm

```
import matplotlib.pyplot as plt
import numpy

class Perceptron(object):

    def __init__(self, dimension,threshold = 10000, learning_rate = 0.01):
        self.dimension = dimension
        self.thresh = threshold
        self.learn = learning_rate
        self.ws = numpy.zeros(dimension + 1)
        self.count = 0
        self.e_in = []


    def predictor(self, point):
        w12 = self.ws[1:]
        xy = point[:self.dimension]
        net = self.ws[0] + numpy.dot(w12, xy)
        if net >= 0:
            output = 1
        else:
            output = -1
        return output

    # function to find e_in -- number of misclassified points
    def misclassified(self, inputs, num = 0):
        for point in inputs:
            label = point[self.dimension]
            xy = point[:self.dimension]
            pred = self.predictor(xy)
            if pred != point[self.dimension]:
                num += 1
        return num

    # stop if you dont find a misclassified point
    def training(self, inputs):
        misclassified = 100 # make this size
        while misclassified != 0:
            if self.count == self.thresh:
                break
            index = numpy.random.randint(low = 0, high = 100)
            point = inputs[index]
            label = point[self.dimension]
            xy = point[:self.dimension]
            pred = self.predictor(xy)
            if pred == point[self.dimension]:
                misclassified -= 1
            else:
                wrong = self.misclassified(inputs)
                self.e_in.append(wrong/100)
                self.ws[1:] += self.learn * (label - pred) * xy
                self.ws[0] += self.learn * (label - pred)
                self.count += 1
                misclassified = 100
```

Samantha Rothman

Code for question 1

```
t = numpy.arange(0,100)
fig, (fig1, fig2) = plt.subplots(1, 2)

# random target line and random data
target = 0.5 * t + 3
data = numpy.random.randint(low = 0, high = 101, size =(100, 3))

# assign labels to data
for i in data:
    x = i[0]
    if i[1] >= (0.5*x+3):
        i[2] = 1
    else:
        i[2] = -1

# generate noise
for x in range(int(len(data)*.05)):
        i = numpy.random.randint(len(data)-1)
        data[i][2] = (data[i][2] * -1)

# compute lin reg
matrix = data[:,0:2]
yvector = data[:, 2:3]

Xdag = numpy.matmul( numpy.linalg.pinv(numpy.matmul(matrix.transpose(), matrix)), matrix.transpose() )
w = numpy.matmul(Xdag, yvector)

# calculate in sample error for lin reg
e_lin = numpy.matmul(matrix, w) - yvector
print("E_in for linear regression: ", numpy.mean(e_lin))

# run perceptron on the data
perceptron =  Perceptron(2) #dimension here
perceptron.training(data)
print("final weights:", perceptron.ws)
print("number of updates:", perceptron.count)

y = -(perceptron.ws[0]/ perceptron.ws[2]) - ((perceptron.ws[1]/perceptron.ws[2]) * t)

# plot the perceptron line and target
fig1.plot(t,target, color = 'g', label = 'f')
fig1.plot(t, y, color = 'm', label = 'g')
fig1.set_ylim([0, 100])

# plot the data points
for i in data:
    if i[2] == -1:
        fig1.scatter(i[0], i[1], marker='_', color = 'r')
    else:
        fig1.scatter(i[0], i[1], marker='+', color = 'b')

# plot the perceptron e_in as a function of iteration
iterations = numpy.arange(0,perceptron.count)
fig2.plot(iterations, perceptron.e_in)

plt.show()
```

Samantha Rothman

Code for question 2 part a

```
#run perceptron on the data
perceptron =  Perceptron(2) #dimension here
perceptron.training(X)
print("final weights:", perceptron.ws)
print("number of updates:", perceptron.count)

# e_in for perceptron
iterations = numpy.arange(0,perceptron.count)
plt.plot(iterations, perceptron.e_in)


# plot perceptron
fig, (fig1, fig2) = plt.subplots(1, 2)
t = numpy.arange(0,1.5)

for i in X:
    if i[2] == -1:
        fig1.scatter(i[0], i[1], marker='_', color = 'r')
    else:
        fig1.scatter(i[0], i[1], marker='+', color = 'b')

y = -(perceptron.ws[0]/ perceptron.ws[2]) - ((perceptron.ws[1]/perceptron.ws[2]) * t)

fig1.plot(t, y, color = 'm', label = 'g')
fig1.set_ylim([0, 1])

#compute lin reg
matrix = X[:,0:2]
yvector = X[:, 2:3]

Xdag = numpy.matmul( numpy.linalg.pinv(numpy.matmul(matrix.transpose(), matrix)), matrix.transpose() )
w = numpy.matmul(Xdag, yvector)

#calculate error for lin reg
e_lin = numpy.matmul(matrix, w) - yvector
print("Error for linear regression: ", numpy.mean(e_lin))

iterations = numpy.arange(0,perceptron.count)
fig2.plot(iterations, perceptron.e_in)


plt.show()
```

Code for question 2 part d

```
# set up 2 dimensional data
X = numpy.random.uniform(size =(100, 3)) #change for dimension

# map data to 5 dimensions
Z = numpy.empty((100,6))

for index in range(100):
    x1 = X[index][0]
    x2 = X[index][1]
    s = [x1, x2, numpy.square(x1), numpy.square(x2), x1*x2, 0]
    Z[index] = s

# assign labels
for i in range(100):
    x = X[i][0]
    y = X[i][1]
    z = Z[i][4]
    label = 1 + 3*x + 4*y + 4*(x**2) + 2*(y**2)
    flag = numpy.random.binomial(1,0.5)
    if  flag == 0 and label > z :
        X[i][-1] = 1
        Z[i][-1] = 1
    else:
        X[i][-1] = -1
        Z[i][-1] = -1


#run perceptron on the data
perceptron =  Perceptron(5) #dimension here
perceptron.training(Z)
print("final weights:", perceptron.ws)
print("number of updates:", perceptron.count)

# e_in for perceptron
iterations = numpy.arange(0,perceptron.count)
plt.plot(iterations, perceptron.e_in)
```