1

(1) $\vec{x} = \begin{pmatrix} -1 \\ -5 \\ -3 \end{pmatrix}$

L1 norm

$||\vec{x}||_1 = 1+5+3 = \underline{\underline{9}}$

$||\vec{x}||_2 = \sqrt{1+5^2+9^2} = \underline{\underline{\sqrt{35}}}$

(2) $\vec{y} = \begin{pmatrix} 0 \\ 4 \\ 16 \end{pmatrix}$

$||\vec{y}||_1 = 0+4+16 = \underline{\underline{20}}$

$||\vec{y}||_2 = \sqrt{0+4^2+16^2} = \underline{\underline{\sqrt{272}}}$

---

2 Cosine values between vectors.

(1) $\vec{x} \begin{pmatrix} 4 \\ -5 \\ 3 \end{pmatrix}, \vec{y} \begin{pmatrix} 8 \\ 10 \\ 6 \end{pmatrix}$

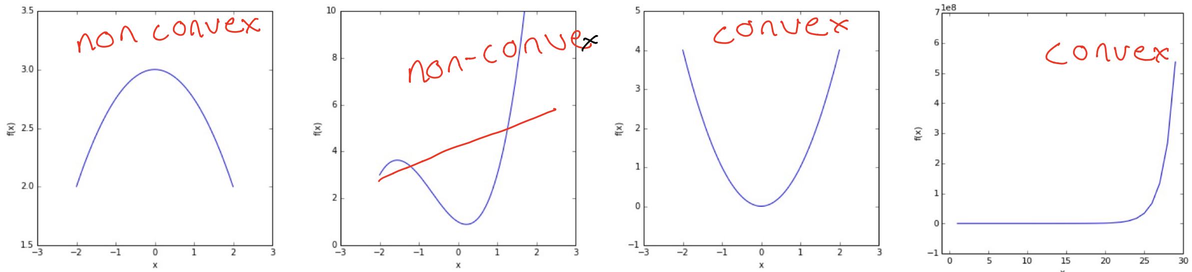
$$\vec{x} \cdot \vec{y} = |\vec{x}| \cdot |\vec{y}| \cos\theta$$

$$\cos\theta = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}|} = \frac{0}{\sim} = \underline{\underline{0}}$$

(2) $\vec{x} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, \quad \vec{y} = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}$

$$\cos\theta = \frac{-1}{\sqrt{3} \cdot \sqrt{3}} = \underline{\underline{-\frac{1}{3}}}$$

3



(a) (b) (c) (d)

(a) & (b) are not convex

(c) & (D) are convex

COGS 181 - Homework 2

4 - Error Metrics

1)

The data matrix X is given by

$$\begin{pmatrix} 8 & 8 & 16 & 4 \\ 4 & 1 & 1 & 16 \\ 6 & 4 & 4 & 2 \\ 4 & 2 & 4 & 1 \\ 8 & 4 & 8 & 2 \end{pmatrix}$$

The label matrix Y is given by $\begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{pmatrix}$

2)

The values computed for y_i is: $\begin{pmatrix} 2.4 \\ -1.5 \\ -1.4 \\ -8.5 \\ 3 \end{pmatrix}$

and the corresponding prediction labels are: $\begin{pmatrix} 1 \\ -1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$

3)

Accuracy: 0.8

Precision: 1.0

Recall: 0.666666666667

F-Score: 0.8

```
import numpy as np


X = np.matrix([
    [8, 1, 16, 4],
    [4, 1, 1, 16],
    [6, 4, 4, 2],
    [4, 2, 4, 1],
    [8, 4, 8, 2]
])
Y = [1,-1,1,-1,1]
W = np.array([1.2, 2, 0.5, 0.7])
b = -20

def classify(X,W,b):
    values = np.inner(W,X) + b
    print "Values:", values
    result = [-1 + (2 * (x >= 0)) for x in values]
    return np.asarray(result)[0][0]

result = classify(X,W,b)
print "Predictions:", result
```

```
Values: [[ 2.4 -1.5 -1.4 -8.5  3. ]]
Predictions: [ 1 -1 -1 -1  1]
```

```
correct_hit = sum([result[i] == Y[i] for i in range(len(result))])
targets_hit = np.sum([(Y[i] == 1 and result[i] == 1) for i in range(len(result))])
total_hits = np.sum([result[i] == 1 for i in range(len(result))])
total_targets = np.sum([Y[i] == 1 for i in range(len(result))])

accuracy = correct_hit / float(len(Y))
precision = targets_hit / float(total_hits)
recall = targets_hit / float(total_targets)
f_score = 2 * precision * recall / ( precision + recall)

print "Accuracy:", accuracy
print "Precision:", precision
print "Recall:", recall
print "F-Score:", f_score
```

```
Accuracy: 0.8
Precision: 1.0
Recall: 0.666666666667
F-Score: 0.8
```

5 - Polynomial Regression

1)

The L2 loss formula is

$$\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

Where

$$f(x_i) = ax_i^2 + bx + c$$

Which can be written in matrix form as:

$$f(X) = X^T W$$

where

$$X = (1, x, x^2)$$

The LSE can be written in matrix form as:

$$G = (X^T W - Y)^T (X^T W - Y)$$

$$G = W^T X^T X W - W^T X^T Y - Y^T X W + Y^T Y$$

We derive on respect with W and set it to 0.

$$\frac{dG}{dW} = X^T X W - X^T X W - X^T Y - X^T Y = 0$$

$$X^T X W = X^T Y$$

Then W will be:

$$W = (X^T X)^{-1} X^T Y$$

2)

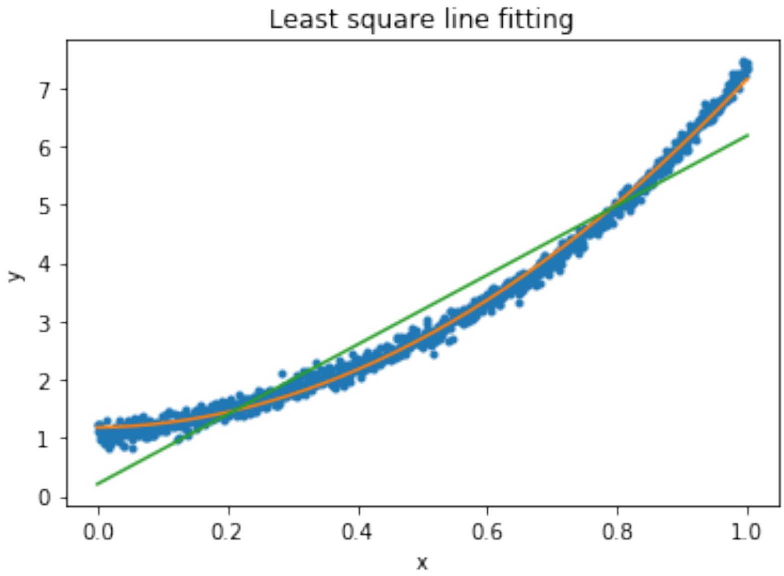
Look at code under, together with a graph. Quadratic model is shown as the orange line.

3)

By just viewing the two models in the plot, we clearly see that the quadratic model is a better solution. This is also confirmed after computing the least square error, where we see that the LSE for linear model is 206, while only 15 for the quadratic model.

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

data = np.loadtxt('data.txt', dtype='float')
%matplotlib inline
x = data[:,0].reshape(len(data),1)
Y = data[:,1].reshape(len(data),1)
# Squared x_i = [1, x, x^2]
X_square = np.hstack((np.ones((len(x),1)),np.power(x,1), np.power(x,2)))
# Linear x_i = [1, x]
X_1 =  np.hstack((np.ones((len(x),1)),np.power(x,1)))

# The LSE for the squared model
sol = np.dot(np.linalg.inv(np.dot(X_square.T,X_square)), np.dot(X_square.T, Y))
# LSE for linear model
sol_hw1 = np.dot(np.linalg.inv(np.dot(X_1.T,X_1)), np.dot(X_1.T, Y))



data_line = plt.plot(x,Y)
plt.hold(True)
quad_line = plt.plot(x,sol[0]+sol[1]*x + sol[2]*x**2, label="Line 1")

lin_line = plt.plot(x,sol_hw1[0] + sol_hw1[1]*x)
plt.title('Least square line fitting')
plt.xlabel('x')
plt.ylabel('y')

plt.show()
def LSE(X,Y,W):
    part1 = (np.dot(X, W) - Y)
    part2 = (np.dot(X, W) - Y)
    return np.dot(part1.T, part2)[0][0]
print "LSE for linear model", LSE(X_1, Y, sol_hw1)
print "LSE for quadratic model:", LSE(X_square, Y,sol,)
```

Green line = Linear model



Orange line = Quadratic model

```
LSE for linear model 206.011861905
LSE for quadratic model: 15.7605107198
```

6 - L1 Loss Regression

1)

L1 loss is given by

$$L(W) = \sum_{i=1}^{n} |(X_i^T W - Y_i)|$$

We want to find

$$W^* = argmin_W \sum_{i=1}^{n} |(X_i^T W - Y_i)|$$

The derivative of f(x) can be written as

$$\frac{|df(W)|}{dW} = sign(X_i^T W - Y_i)X_i$$

Then we have

$$\frac{dL(W)}{dW} = \sum_{i=1}^{n} sign(X_i^T W - Y_i)X_i$$

And then we can find W by:

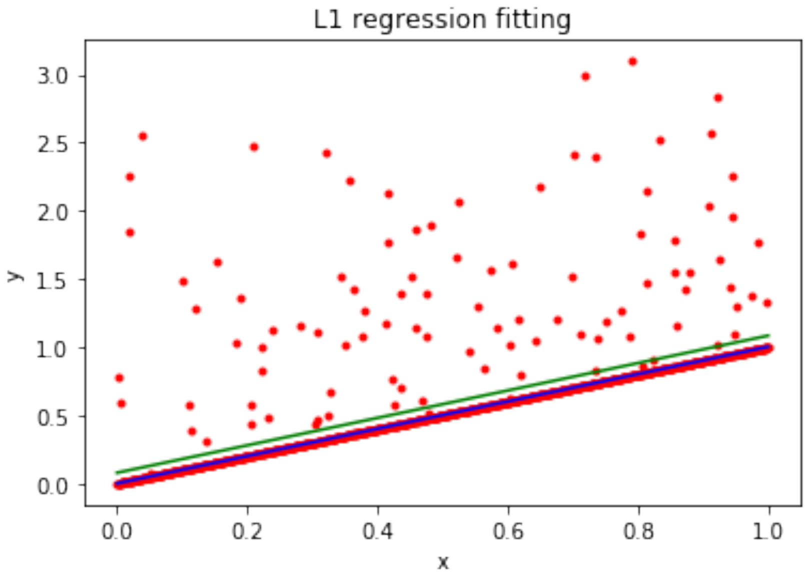
$$W_{t+1} = W_t - \lambda_t \frac{dL(W)}{dW}$$

2) & 3)

Given below

```
%matplotlib inline
alpha = 0.00001
iterations = 2000
W = [0, 0]

data = np.loadtxt('data_2.txt', dtype='float')

x = data[:,0].reshape(len(data),1)
Y = data[:,1].reshape(len(data),1)
# Linear x_i = [1, x]
X =  np.hstack((np.ones((len(x),1)),np.power(x,1)))

def L_prime(X,Y,W):
    res = 0
    for i in range(len(X)):
        sign = np.sign(np.dot(X[i], W) - Y[i])
        res += sign * X[i]
    return res

def regression(X, Y, alpha, iterations):
    W = np.array([0, 0]).T
    for i in range(iterations):
        W = W - alpha * L_prime(X ,Y, W)
    return W
# The L1 error regression
sol_6 = regression(X,Y,alpha,iterations)
# L2 error regression
sol_hw1 = np.dot(np.linalg.inv(np.dot(X.T,X)), np.dot(X.T, Y))

data_line = plt.plot(x,Y)
plt.hold(True)
plt.plot(x,sol_6[0] + sol_6[1]*x)
plt.plot(x,sol_hw1[0] + sol_hw1[1]*x)
plt.title('L1 regression fitting')
plt.xlabel('x')
plt.ylabel('y')
```

Green line = LSE regression



Blue line = L1 Error norm regression