Basic Machine Learning Math

Introduction

In this exercise we are to show an understanding for some of the mathematical building bricks of machine learning. This includes working with matrices and arrays both mathematically and pythonic.

Vector and matrix representation in python

Qa Given the following $\mathbf{x}^{(i)}$'s, construct and print the $\mathbf{X}$ matrix in python.

$$
\begin{aligned}
\mathbf{x}^{(1)} &= [\,1, 2, 3\,]^T \\
\mathbf{x}^{(2)} &= [\,4, 2, 1\,]^T \\
\mathbf{x}^{(3)} &= [\,3, 8, 5\,]^T \\
\mathbf{x}^{(4)} &= [\,-9, -1, 0\,]^T
\end{aligned}
$$

Use *numpy* as matrix container, do not use the built-in python lists or the numpy matrix subclass.

Qa Implementation

In [5]:

```
import numpy as np

y = np.array([1,2,3,4]) # NOTE:  you'll need this later


# we create 4 vectors (or arrays in python to describe x1-4)

x1 = np.array([1, 2, 3]).T
x2 = np.array([4, 2, 1]).T
x3 = np.array([3, 8, 5]).T
x4 = np.array([-9, -1, 0]).T

# using the vectors x1-4 we create the full matrix X

X = np.array([x1.T, x2.T, x3.T, x4.T])

print(X)
```

```
[[ 1  2  3]
 [ 4  2  1]
 [ 3  8  5]
 [-9 -1  0]]
```

Qa Result

As seen on the prinout above X is a nice 3x4 matrix, just as it ought to be

Norms, metrics or distances

$$\mathcal{L}_2: \quad ||\mathbf{x}||_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$$

$$\mathcal{L}_1: \quad ||\mathbf{x}||_1 = \sum_{i} |x_i|$$

Qb Implement the $\mathcal{L}_1$ and $\mathcal{L}_2$ norms for vectors in python.

First, do not use any built-in methods, not even x.dot(). Name your functions L1 and L2 respectively, they both take one vector as input argument.

But test your implementation against some built-in functions, say `numpy.linalg.norm`

When this works, and passes the tests below, optimize the $\mathcal{L}_2$, such that it uses np.numpy's dot operator instead of an explicit sum, call this function L2Dot. This implementation must be pythonic, i.e. it must not contain explicit for- or while-loops.

Qb Implementation

Implement the L1, L2 and L2Dot functions

In [20]:

```
# L1 is solved using a for loop to sum all the values in the array
def L1(xi):

    assert xi.shape[0]>=0, "L1 xi matrix is empty"

    sum = 0
    xi = abs(xi)

    for x in xi:
        sum += x

    return sum

# L2 is solved using a for loop to sum all the values, squared, in the array. The root
 value of the sum is then returned
def L2(xi):

    assert xi.shape[0]>=0, "L2 xi matrix is empty"

    sum = 0
    xi = abs(xi)

    for x in xi:
        sum += x**2

    return sum**0.5

# L2Dot is solved using pythons inbuilt sum function.
def L2Dot(xi):
    assert xi.shape[0]>=0, "L2Dot xi matrix is empty"

    return sum(abs(xi)**2)**0.5


# TEST vectors: here I test your implementation...calling your L1() and L2() functions
tx=np.array([1, 2, 3, -1])
ty=np.array([3,-1, 4,  1])

expected_d1=8.0
expected_d2=4.242640687119285

d1=L1(tx-ty)
d2=L2(tx-ty)

print("tx-ty=",tx-ty,", d1-expected_d1=",d1-expected_d1,", d2-expected_d1=",d2-expected
_d2)

eps=1E-9
assert np.fabs(d1-expected_d1)<eps, "L1 dist seems to be wrong"
assert np.fabs(d2-expected_d2)<eps, "L2 dist seems to be wrong"

# comment-in once your L2Dot fun is ready...
d2dot=L2Dot(tx-ty)
print("d2dot-expected_d2=",d2dot-expected_d2)
assert np.fabs(d2dot-expected_d2)<eps, "L2Ddot dist seem to be wrong"
```

```
tx-ty= [-2  3 -1 -2] , d1-expected_d1= 0.0 , d2-expected_d1= 0.0
d2dot-expected_d2= 0.0
```

Qb Result

In the prinouts we see that the results from d1, d2 and d2dot matches the expected values, hence the implementation was done correct.

Performance metrics

$$\begin{aligned} \text{MSE} &= \frac{1}{n}\sum_{i=1}^{n}(\hat{Z}_i - Z_i)^2 = \frac{1}{n}SS \\ \text{RMSE} &= \sqrt{MSE} \end{aligned}$$

Qc Construct the Root Mean Square Error (RMSE) function (Equation 2-1 [HOLM]).

Call the function RMSE, and evaluate it using the $\mathbf{X}$ matrix and $\mathbf{y}$ from Qa

We implement a dummy hypothesis function, that just takes the first column of $\mathbf{X}$ as its 'prediction'

$$h_{dummy}(\mathbf{X}) = \mathbf{X}(:,0)$$

Qc Implementation

Implement an RMSE function

In [12]:

```
# the RMSE function is implemented using the mean function since the sum part is just t
he mean of the array (predicted - expected)
# a check for an empty list is done
def RMSE(X,y):

    #check for empty list
    assert len(X) != 0, "m is 0"

    return np.sqrt(((X-y)**2).mean())


# TEST vector:
def h(X):
    if X.ndim!=2:
        raise ValueError("excpeted X to be of ndim=2, got ndim=",X.ndim)
    if X.shape[0]==0 or X.shape[1]==0:
        raise ValueError("X got zero data along the 0/1 axis, cannot continue")
    return X[:,0]

eps=1E-9

print(h(X))
r=RMSE(h(X),y)
expected=6.57647321898295
print("RMSE=",r,", diff=",r-expected)
assert r-expected<eps, "your RMSE dist seems to be wrong"
```

```
[ 1  4  3 -9]
RMSE= 6.576473218982953 , diff= 2.6645352591003757e-15
```

Qc Results

As seen in the printouts the results does not differ from the extected values.

MAE

Qd Similar construct the Mean Absolute Error (MAE) function (Equation 2-2 [HOLM]) and evaluate it.

The MAE will algorithmic wise be similar to the MSE part from using the $\mathcal{L}_1$ instead of the $\mathcal{L}_2$ norm.

Qd Implementation

Implement a MSE function

In [11]:

```
# like the RMSE function the mean is again taking, however here it is shown on a more m
athematically form using sum and 1/m
def MAE(X,y):

    #check for empty list
    m = len(X)
    assert m != 0, "m is 0"

    return 1/m*sum(abs(X-y))


# Qd Results

# TEST vector:
r=MAE(h(X), y)
expected=3.75
print("MAE=",r,", diff=",r-expected)
assert r-expected<eps, "MAE dist seems to be wrong"
```

```
MAE= 3.75 , diff= 0.0
```

Qd Result

Again the results matches the expected values

Pythonic Code

Qe Robust Code

Add error checking code (asserts or exceptions), that checks for right $\hat{\mathbf{y}}$-$\mathbf{y}$ sizes of the RMAE and MEA functions.

Also add error checking to all you previously tested L2() and L1() functions, and re-run all your tests.

In [2]:

```
# TODO: solve Qe...you need to modify your python cells above

# check asserts throughout the code above.
```