```
import numpy as np
```

**Name: Sudhir Sharma**

**ID NO 12041500**

**Branch CSE**

**CS 550 Homework 1**

**PART A ****
**ans**
**** a)** As the pricing equation is given as : P = C*max {L, 20} + 100
Where staring value of c is 20 and L : Length of the journey (kms) Using The given equation the minimum price of a ticket using the starting values (ie 20) comes
out to be 500.

**b) **

If user accept's the ride then value of C will increase by 0.01*[max(l,20)] to max. the profit and if its rejected then C will decrease by 0.01*[max(l,20)].

In the event that there is demand, the update rule tries to capitalise on it, and it makes up for poor demand by lowering the price proportionate to the existing price. Let's change it this way.

if the user accepts the rides at the quoted price:

```
C = C + scaling factor *∇cP/P
```

if the user rejects the ride at the quoted price

```
C = max(C - scaling factor *∇cP/P,threshold)
```

Although the scaling factor is set here to 0.01 as its initial value, it can be adjusted to any number to match the beginning value. The lowest value of C that the firm can afford is called the threshold. With this improvement, the spike rate is considerably lower than it was before.

**c**)

```
C = 5
train = [
    {'l':10, 'Accepted':'YES'},
    {'l':20, 'Accepted':'YES'},
    {'l':10, 'Accepted':'YES'},
    {'l':15, 'Accepted':'NO'},
    {'l':35, 'Accepted':'YES'},
    {'l':30, 'Accepted':'NO'},
]

for i in train:
   l = i['l']
   v = i['Accepted']
   print("C before :",C)
   print("Price :",C*max(l,20) + 100)
   if v == "YES":
      C = C + 0.01*max(l, 20)
   else:
      C = C - 0.01*max(l, 20)
   print("Updated value Of C: ",C)
   print()
```

```
C before : 5
Price : 200
Updated value Of C:  5.2

C before : 5.2
Price : 204.0
Updated value Of C:  5.4

C before : 5.4
Price : 208.0
Updated value Of C:  5.6000000000000005

C before : 5.6000000000000005
Price : 212.0
Updated value Of C:  5.4

C before : 5.4
Price : 289.0
Updated value Of C:  5.75

C before : 5.75
Price : 272.5
Updated value Of C:  5.45
```

| L | Price Quoted | User Purchases? | C |
|---|---|---|---|
| 10 | 200 | Yes | 5.2 |
| 20 | 204 | Yes | 5.4 |
| 10 | 208 | Yes | 5.6 |
| 15 | 212 | No | 5.4 |
| 35 | 289 | Yes | 5.75 |
| 30 | 272 | No | 5.45 |

**PART B**

**I** a. The training set is for model training and the test set is for model validation.

**Explaination**

The reason is that when the dataset is split into train and test sets, there will not be enough data in the training dataset for the model to learn an effective mapping of inputs to outputs. There will also not be enough data in the test set to effectively evaluate the model performance. So, We train the model firstly ,then ascertain if its working properly on the test set.

**II** c. Traditional computing problems such as inverting matrices, multiplying numbers etc

**Explaination**

Traditional issues can't be greatly enhanced using Machine Learning.

**III** a. one training sample

**Explaination**

Stochastic gradient method uses one sample at each iteration, giving it faster convergence and less accuracy.

An important parameter of Gradient Descent (GD) is the size of the steps, determined by the learning rate hyperparameters. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time, and if it is too high we may jump the optimal value.

**IV** Third image depicts underfitting.

**V** Gradient Descent works well for large number of features, because it uses O(n) time complexity.

We can also use batch gradient descent, stochastic gradient descent, or mini-batch gradient descent. SGD and MBGD would work the best because neither of them need to load the entire dataset into memory in order to take 1 step of gradient descent. Batch would be ok with the caveat that you have enough memory to load all the data.

The normal equations method would not be a good choice because it is computationally inefficient. The main cause of the computational complexity comes from inverse operation on an (n x n) matrix.

O n2 . 4 to O n3

**VI** Gradient Descent algorithms suffer most by outliers, since they are overfitting by default. We can apply feature scaling to converge quickly.

**VII** Gradient descent produces a convex shaped graph which only has one global optimum. Therefore, it cannot get stuck in a local minimum.

**VIII** When the training error and validation error are close to each other and high that means your model is underfitting (i.e. it has high bias). You should try to reduce the regularization hyperparameter.

**IX** No. The issue is that stochastic gradient descent and mini-batch gradient descent have randomness built into them. This means that they can find their way to nearby the global optimum, but they generally don't converge. One way to help them converge is to gradually reduce the learning rate hyperparameter.

**X** When there are many features and few of them can be dropped because of their low influence on target. It simplifies our model

**PART C**

The distance function I'm using is euclidean distance

distance = math.sqrt(((x2 - x1) ** 2) + (y2 - y1) ** 2)

```
data = [
    {"label": "Normal", "image": np.array([[2, 1, 2], [1, 2, 1], [2, 1, 1]])},
    {"label": "A", "image": np.array([[4, 3, 2], [3, 3, 1], [2, 1, 1]])},
    {"label": "B", "image": np.array([[2, 1, 2], [1, 2, 3], [2, 3, 4]])},
    {"label": "C", "image": np.array([[2, 3, 4], [1, 3, 3], [2, 1, 1]])},
]


test = [
    np.array([
        [1, 2, 2],
        [1, 2, 1],
        [1, 1, 1]
    ]),
    np.array([
        [1, 4, 4],
        [1, 3, 4],
        [2, 2, 1]
    ])
]
```

After calculating for each test

distance(Normal,t1)= 3

distance(A,t1)= 16

distance(B,t1)= 20

distance(C,t1)= 12

distance(Normal,t2)= 25

distance(A,t2)= 28

distance(B,t2)= 26

distance(C,t2)= 4

So t1 is normal and t2 is C.

**PART D**

$$w^{(t+1)} = w^t + \eta_t \sum_{n=1}^{N} 2(y_n - w^{t^T} x_n)x_n$$

Assuming N=1, GD will be

$$w^{(t+1)} = w^t + \eta_t 2(y - w^{t^T} x)x$$

Taking Transpose both side

$$w^{(t+1)^T} x = w^{t^T} x + \eta_t 2(y - x^T w^t)x^T$$

$$y - w^{(t+1)^T} x = (y - w^{t^T} x) - \eta_t 2(y - x^T w^t)x^T x$$

as

$$x^T x >= 0.x^T w^t = wt^T x$$

if

$$((y - w^{t^T} x) == ispositive$$

$$y - w^{t+1^T} x < ((y - w^{t^T} x) - (+ve)$$

else if

$$((y - w^{t^T} x) == isnegative$$

$$y - w^{t+1^T} x = ((y - w^{t^T} x) - (-ve)$$

$$|y - w^{t+1^T} x| < ((y - w^{t^T} x)$$

So

Assuming $N$ =1,

Hence shown that GD update improves prediction on the training input ($\boldsymbol{x}n$, $yn$), i.e, $yn$ is closer to $\boldsymbol{w}(t$+1)⊤$\boldsymbol{x}n$ than to $\boldsymbol{w}(t)$⊤$\boldsymbol{x}n$

✓ 0s completed at 11:07 PM ● ✕