CSCE 5218 / CSCE 4930 Deep Learning

HW1a The Perceptron (20 pt)

In [7]:
```
# Get the datasets
test = !wget http://huang.eng.unt.edu/CSCE-5218/test.dat
train = !wget http://huang.eng.unt.edu/CSCE-5218/train.dat
```

In [8]:
```
# Take a peek at the datasets
!head train.dat
!head test.dat
```

```
A1      A2      A3      A4      A5      A6      A7      A8      A9      A10     A11
A12     A13
1       1       0       0       0       0       0       0       1       1       0
0       1       0
0       0       1       1       0       1       1       0       0       0       0
0       1       0
0       1       0       1       1       0       1       0       1       1       1
0       1       1
0       0       1       0       0       1       0       1       0       1       1
1       1       0
0       1       0       0       0       0       0       1       1       1       1
1       1       0
0       1       1       1       0       0       0       1       0       1       1
0       1       1
0       1       1       0       0       0       1       0       0       0       0
0       1       0
0       0       0       1       1       0       1       1       1       0       0
0       1       0
0       0       0       0       0       0       1       0       1       0       1
0       1       0
A1      A2      A3      A4      A5      A6      A7      A8      A9      A10     A11
A12     A13
1       1       1       1       0       0       1       1       0       0       0
1       1       0
0       0       0       1       0       0       1       1       0       1       0
0       1       0
0       1       1       1       0       1       1       1       1       0       0
0       1       0
0       1       1       0       1       0       1       1       1       0       1
0       1       0
0       1       0       0       0       1       0       1       0       1       0
0       1       0
0       1       1       0       0       1       1       1       1       1       1
0       1       0
0       1       1       1       0       0       1       1       0       0       0
1       1       0
0       1       0       0       1       0       0       1       1       0       1
1       1       0
1       1       1       1       0       0       1       1       0       0       0
0       1       0
```

Out[8]:
```
["'wget' is not recognized as an internal or external command,",
 'operable program or batch file.']
```

Build the Perceptron Model

You will need to complete some of the function definitions below. DO NOT import any other libraries to complete this.

In [5]:

```
import math
import itertools
import re


# Corpus reader, all columns but the last one are coordinates;
#   the last column is the label
def read_data(file_name):
    f = open(file_name, 'r')

    data = []
    # Discard header line
    f.readline()
    for instance in f.readlines():
        if not re.search('\t', instance): continue
        instance = list(map(int, instance.strip().split('\t')))
        # Add a dummy input so that w0 becomes the bias
        instance = [-1] + instance
        data += [instance]
    return data


def dot_product(array1, array2):
    dot_product=sum([k*l for(k,l) in zip(array1,array2)])
    return dot_product


def sigmoid(x):
    sig = 1/(1 +math.exp(-x))
    return sig

# The output of the model, which for the perceptron is
# the sigmoid function applied to the dot product of
# the instance and the weights
def output(weight, instance):
    values = sigmoid(dot_product(weight,instance))
    return values

# Predict the label of an instance; this is the definition of the perceptron
# you should output 1 if the output is >= 0.5 else output 0
def predict(weights, instance):
    if output(weights,instance) >= 0.5:
        return 1
    else:
        return 0


# Accuracy = percent of correct predictions
def get_accuracy(weights, instances):
    # You do not to write code like this, but get used to it
    correct = sum([1 if predict(weights, instance) == instance[-1] else 0
                   for instance in instances])
    return correct * 100 / len(instances)


# Train a perceptron with instances and hyperparameters:
#       lr (learning rate)
#       epochs
# The implementation comes from the definition of the perceptron
#
# Training consists on fitting the parameters which are the weights
# that's the only thing training is responsible to fit
# (recall that w0 is the bias, and w1..wn are the weights for each coordinate)
```

```
#
# Hyperparameters (lr and epochs) are given to the training algorithm
# We are updating weights in the opposite direction of the gradient of the error,
# so with a "decent" lr we are guaranteed to reduce the error after each iteration.
def train_perceptron(instances, lr, epochs):

    #TODO: name this step
    weights = [0] * (len(instances[0])-1)

    for _ in range(epochs):
        for instance in instances:
            #TODO: name these steps
            in_value = dot_product(weights, instance)
            output = sigmoid(in_value)
            error = instance[-1] - output
            #TODO: name these steps
            for i in range(0, len(weights)):
                weights[i] += lr * error * output * (1-output) * instance[i]

    return weights
```

Run it

In [6]:

```
instances_tr = read_data("train.dat")
instances_te = read_data("test.dat")
lr = 0.005
epochs = 5
weights = train_perceptron(instances_tr, lr, epochs)
accuracy = get_accuracy(weights, instances_te)
print(f"#tr: {len(instances_tr):3}, epochs: {epochs:3}, learning rate: {lr:.3f}; "
      f"Accuracy (test, {len(instances_te)} instances): {accuracy:.1f}")
```

```
#tr: 400, epochs:   5, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
```

Questions

Answer the following questions. Include your implementation and the output for each question.

Question 1

In `train_perceptron(instances, lr, epochs)` , we have the follosing code:

```
in_value = dot_product(weights, instance)
output = sigmoid(in_value)
error = instance[-1] - output
```

Why don't we have the following code snippet instead?

```
output = predict(weights, instance)
error = instance[-1] - output
```

TODO Add your answer here (text only)

Answer: Above snippet code helps to get better precision and accuracy. sigmoid activation function accept the values between 0 and 1 and based on set of threshold value it performs classification. Here we have used the sigmoid activation function to achieve better perceptron

model with good prediction. By using sigmoid fuction we are non-linear activation function instead of binary or linear function. when we use inary or linear function, It only accept either 0 or 1 ut using sigmoid we have chnace to get all values between 0 to 1.

Question 2

Train the perceptron with the following hyperparameters and calculate the accuracy with the test dataset.

```
tr_percent = [5, 10, 25, 50, 75, 100] # percent of the training
dataset to train with
num_epochs = [5, 10, 20, 50, 100]              # number of epochs
lr = [0.005, 0.01, 0.05]              # learning rate
```

TODO: Write your code below and include the output at the end of each training loop (NOT AFTER EACH EPOCH) of your code.The output should look like the following:

```
# tr:  20, epochs:   5, learning rate: 0.005; Accuracy (test, 100
instances): 68.0
# tr:  20, epochs:  10, learning rate: 0.005; Accuracy (test, 100
instances): 68.0
# tr:  20, epochs:  20, learning rate: 0.005; Accuracy (test, 100
instances): 68.0
[and so on for all the combinations]
```

You will get different results with different hyperparameters.

TODO Add your answer here (code and output in the format above)

Answer:

In [49]:

```
instances_tr = read_data("train.dat")
instances_te = read_data("test.dat")
tr_percent = [5, 10, 25, 50, 75, 100] # percent of the training dataset to train wit
num_epochs = [5, 10, 20, 50, 100]     # number of epochs
lr_array = [0.005, 0.01, 0.05] # learning rate
acc =[]
epo=[]
learn_rate=[]
tr=[]

for lr in lr_array:
  for tr_size in tr_percent:
    for epochs in num_epochs:
      size =  round(len(instances_tr)*tr_size/100)
      pre_instances = instances_tr[0:size]
      weights = train_perceptron(pre_instances, lr, epochs)
      accuracy = get_accuracy(weights, instances_te)
    print(f"#tr: {len(pre_instances):0}, epochs: {epochs:3}, learning rate: {lr:.3f}
           f"Accuracy (test, {len(instances_te)} instances): {accuracy:.1f}")
    acc.append(accuracy)
    epo.append(epochs)
    learn_rate.append(lr)
    tr.append(len(pre_instances))
```

```
#tr: 20, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
#tr: 40, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
```

```
#tr: 100, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 68.0
#tr: 200, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 74.0
#tr: 300, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 78.0
#tr: 400, epochs: 100, learning rate: 0.005; Accuracy (test, 100 instances): 77.0
#tr: 20, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
#tr: 40, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 68.0
#tr: 100, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 71.0
#tr: 200, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 78.0
#tr: 300, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 80.0
#tr: 400, epochs: 100, learning rate: 0.010; Accuracy (test, 100 instances): 80.0
#tr: 20, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 64.0
#tr: 40, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 69.0
#tr: 100, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 77.0
#tr: 200, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 76.0
#tr: 300, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 77.0
#tr: 400, epochs: 100, learning rate: 0.050; Accuracy (test, 100 instances): 80.0
```

Question 3

Write a couple paragraphs interpreting the results with all the combinations of hyperparameters. Drawing a plot will probably help you make a point. In particular, answer the following:

Answer: It can be predict from question 2 output that using various parameters, percent of traning of dataset, epochs, learning rate, tr there are different accuracies. The more percent of data we have provided in the training set the more chance of accuracy we get. -The more the learning rate the better the accuracy we get, so, It is clearly seen from the data that learning rate plays a major role. It is seen that if epoch is high then the model prformance wll increase.It better helps in updates of the weights and helps in acheiving the local minima.

A. Do you need to train with all the training dataset to get the highest accuracy with the test dataset?

Answer A) Yes, we need all dataset of training to get correct and better accuracy. It is obvious that considering every data from the dataset helps in better prediction.

B. How do you justify that training the second run obtains worse accuracy than the first one (despite the second one uses more training data)?

Answer: The answer behind the above question is that in the secod run the learning rate is lower then the first one.

```
#tr: 100, epochs:  20, learning rate: 0.050; Accuracy (test, 100
instances): 71.0
#tr: 200, epochs:  20, learning rate: 0.005; Accuracy (test, 100
instances): 68.0
```

C. Can you get higher accuracy with additional hyperparameters (higher than `80.0`)?

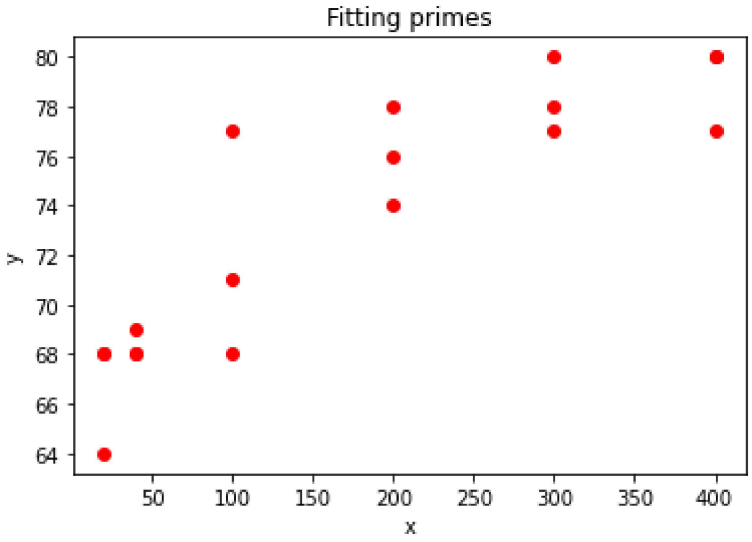
Answer: Yes, there is chance to get high accuracy by extra hyperparameters.

D. Is it always worth training for more epochs (while keeping all other hyperparameters fixed)?

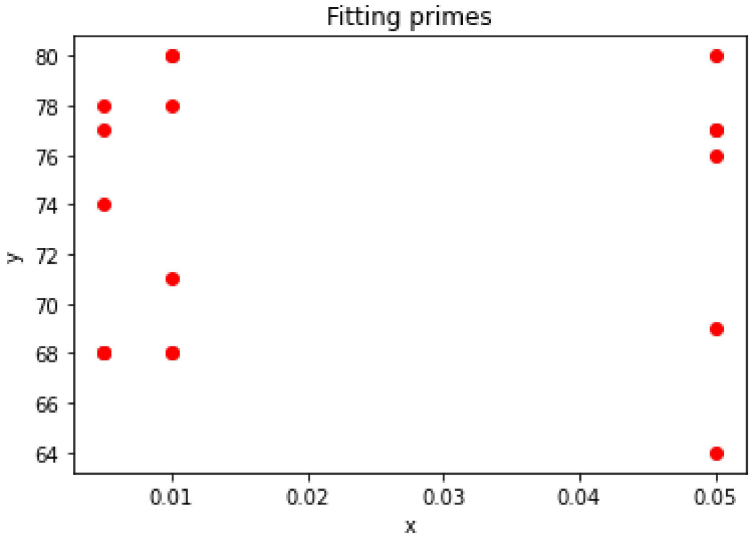
Answer: No, It is not always worth that more epochs effect in the accuracy may be there is chance that with less number of epoch we can get the better accuracy.

In [50]:
```
import matplotlib.pyplot as plt
```

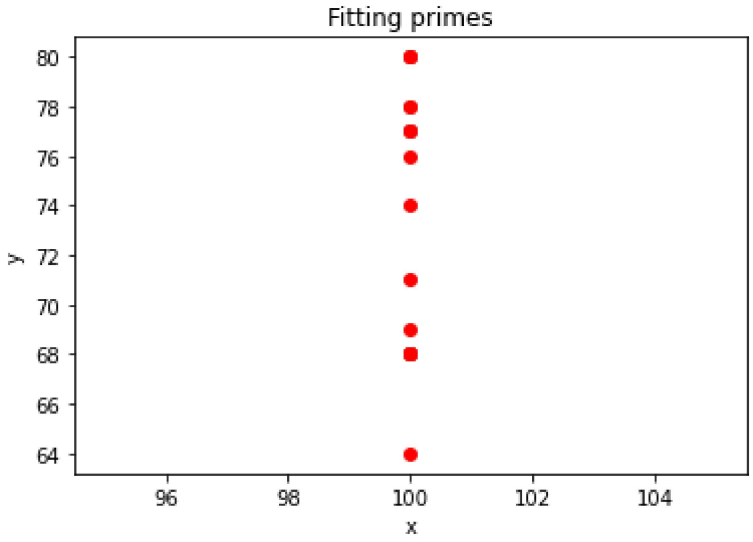
In [58]:
```
plt.scatter(tr, acc, c='r', label='data')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Fitting primes')
plt.show()
```



In [61]:
```
plt.scatter(learn_rate, acc, c='r', label='data')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Fitting primes')
plt.show()
```



In [62]:
```
plt.scatter(epo, acc, c='r', label='data')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Fitting primes')
plt.show()
```


Fitting primes
80
78
76
74
72
70
68
66
64
y
96
98
100
102
104
x