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

```
--2023-02-16 03:11:10--  http://huang.eng.unt.edu/CSCE-5218/train.dat
Resolving huang.eng.unt.edu (huang.eng.unt.edu)... 129.120.123.155
Connecting to huang.eng.unt.edu (huang.eng.unt.edu)|129.120.123.155|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11244 (11K)
Saving to: ‘train.dat’

train.dat           100%[===================>]  10.98K  --.-KB/s    in 0s

2023-02-16 03:11:11 (68.6 MB/s) - ‘train.dat’ saved [11244/11244]

--2023-02-16 03:11:11--  http://huang.eng.unt.edu/CSCE-5218/test.dat
Resolving huang.eng.unt.edu (huang.eng.unt.edu)... 129.120.123.155
Connecting to huang.eng.unt.edu (huang.eng.unt.edu)|129.120.123.155|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2844 (2.8K)
Saving to: ‘test.dat’

test.dat            100%[===================>]   2.78K  --.-KB/s    in 0s

2023-02-16 03:11:11 (242 MB/s) - ‘test.dat’ saved [2844/2844]
```

```
# 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
```

CSCE 5218 / CSCE 4930 Deep Learning

HW1a The Perceptron (20 pt)

```
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):
    # Calculate the dot product of array1 and array2
    return sum([array1[i] * array2[i] for i in range(len(array1))])


def sigmoid(x):
    # Calculate the output of the sigmoid function for x
    return 1 / (1 + math.exp(-x))


# 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):
    in_value = dot_product(weight, instance)
    return sigmoid(in_value)


# 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):
    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):

    # Initialize the weights to 0
    weights = [0] * (len(instances[0])-1)

    for _ in range(epochs):
        for instance in instances:
            # Calculate the input value
            in_value = dot_product(weights, instance)
            # Calculate the output value
            output_value = sigmoid(in_value)
            # Calculate the error
            error = instance[-1] - output_value
            # Update the weights
            for i in range(0, len(weights)):
                weights[i] += lr * error * output_value * (1-output_value) * instance[i]

    return weights
```

Double-click (or enter) to edit

```
# Get the datasets
!wget http://www.cse.unt.edu/~blanco/csce5218/hw1a/train.dat
!wget http://www.cse.unt.edu/~blanco/csce5218/hw1a/test.dat
```

```
--2023-02-16 03:11:12--  http://www.cse.unt.edu/~blanco/csce5218/hw1a/train.dat
Resolving www.cse.unt.edu (www.cse.unt.edu)... 129.120.151.91
Connecting to www.cse.unt.edu (www.cse.unt.edu)|129.120.151.91|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11244 (11K) [application/x-ns-proxy-autoconfig]
Saving to: ‘train.dat.1’

train.dat.1         100%[===================>]  10.98K  --.-KB/s    in 0s

2023-02-16 03:11:12 (182 MB/s) - ‘train.dat.1’ saved [11244/11244]

--2023-02-16 03:11:13--  http://www.cse.unt.edu/~blanco/csce5218/hw1a/test.dat
Resolving www.cse.unt.edu (www.cse.unt.edu)... 129.120.151.91
Connecting to www.cse.unt.edu (www.cse.unt.edu)|129.120.151.91|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2844 (2.8K) [application/x-ns-proxy-autoconfig]
Saving to: ‘test.dat.1’

test.dat.1          100%[===================>]   2.78K  --.-KB/s    in 0s

2023-02-16 03:11:13 (257 MB/s) - ‘test.dat.1’ saved [2844/2844]
```

```
from ast import YieldFrom
instances_tr = read_data("train.dat")
instances_te = read_data("test.dat")

x=[]
for i in range(0,len(instances_tr)-1):
  if(i%2==1):
    x.append(instances_tr[i])
y=[]
for i in range(0,len(instances_te)-1):
  if(i%2==1):
    y.append(instances_te[i])

lr = 0.005
epochs = 5
weights = train_perceptron(x, lr, epochs)

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

```
#tr: 800, epochs:   5, learning rate: 0.005; Accuracy (test, 200 instances): 67.7
```

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)

The reason we use dot_product and sigmoid in the train_perceptron function is that this is the standard way of implementing a perceptron model.

The dot_product function calculates the dot product of the weights and input features, which produces a single value that represents the weighted sum of the input features. This value is then passed through the sigmoid function to produce a prediction value between 0 and 1.

The predict function you suggested is not a standard part of a perceptron implementation because it simply calculates the dot product of the weights and input features, without applying any activation function. This means that the output of predict would not be in the desired range of 0 to 1, which is required for binary classification tasks.

Therefore, using the sigmoid function is necessary in order to produce an output that is within the correct range for binary classification tasks.

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 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 differet hyperparameters.

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

```
instances_tr = read_data("train.dat")
instances_te = read_data("test.dat")
tr_percent = [5, 10, 17, 25,35, 50, 60, 75, 87, 100] # percent of the training dataset to train with
num_epochs = [5, 7, 10, 15, 20, 35, 50, 60,70,80, 90, 100]     # number of epochs
lr_array = [0.005, 0.01, 0.2,0.03,0.04,0.05]        # learning rate

x=[]
for i in range(0,len(instances_tr)-1):
  if(i%2==1):
    x.append(instances_tr[i])
y=[]
for i in range(0,len(instances_te)-1):
  if(i%2==1):
    y.append(instances_te[i])
for lr in lr_array:
  for tr_size in tr_percent:
    for epochs in num_epochs:
      size =  round(len(x)*tr_size/100)
      pre_instances = x[0:size]
      weights = train_perceptron(pre_instances, lr, epochs)
      accuracy = get_accuracy(weights, y)
    print(f"#tr: {len(pre_instances):0}, epochs: {epochs:3}, learning rate: {lr:.3f}; "
            f"Accuracy (test, {len(y)} instances): {accuracy:.1f}")
```

```
#tr: 68, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 67.7
#tr: 100, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 67.7
#tr: 140, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 67.7
#tr: 200, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 70.7
#tr: 239, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 77.8
#tr: 299, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 76.8
#tr: 347, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 75.8
#tr: 399, epochs: 100, learning rate: 0.005; Accuracy (test, 99 instances): 75.8
#tr: 20, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 67.7
#tr: 40, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 67.7
#tr: 68, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 67.7
#tr: 100, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 69.7
#tr: 140, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 73.7
#tr: 200, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 77.8
#tr: 239, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 78.8
#tr: 299, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 78.8
#tr: 347, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 78.8
#tr: 399, epochs: 100, learning rate: 0.010; Accuracy (test, 99 instances): 78.8
#tr: 20, epochs: 100, learning rate: 0.200; Accuracy (test, 99 instances): 63.6
#tr: 40, epochs: 100, learning rate: 0.200; Accuracy (test, 99 instances): 62.6
#tr: 68, epochs: 100, learning rate: 0.200; Accuracy (test, 99 instances): 73.7
#tr: 100, epochs: 100, learning rate: 0.200; Accuracy (test, 99 instances): 73.7
```

```
#tr: 239, epochs: 100, learning rate: 0.200; Accuracy (test, 99 instances): 76.8
#tr: 299, epochs: 100, learning rate: 0.200; Accuracy (test, 99 instances): 76.8
#tr: 347, epochs: 100, learning rate: 0.200; Accuracy (test, 99 instances): 80.8
#tr: 399, epochs: 100, learning rate: 0.200; Accuracy (test, 99 instances): 79.8
#tr: 20, epochs: 100, learning rate: 0.030; Accuracy (test, 99 instances): 67.7
#tr: 40, epochs: 100, learning rate: 0.030; Accuracy (test, 99 instances): 69.7
#tr: 68, epochs: 100, learning rate: 0.030; Accuracy (test, 99 instances): 71.7
#tr: 100, epochs: 100, learning rate: 0.030; Accuracy (test, 99 instances): 71.7
#tr: 140, epochs: 100, learning rate: 0.030; Accuracy (test, 99 instances): 74.7
#tr: 200, epochs: 100, learning rate: 0.030; Accuracy (test, 99 instances): 78.8
#tr: 239, epochs: 100, learning rate: 0.030; Accuracy (test, 99 instances): 76.8
#tr: 299, epochs: 100, learning rate: 0.030; Accuracy (test, 99 instances): 78.8
#tr: 347, epochs: 100, learning rate: 0.030; Accuracy (test, 99 instances): 79.8
#tr: 399, epochs: 100, learning rate: 0.030; Accuracy (test, 99 instances): 79.8
#tr: 20, epochs: 100, learning rate: 0.040; Accuracy (test, 99 instances): 66.7
#tr: 40, epochs: 100, learning rate: 0.040; Accuracy (test, 99 instances): 70.7
#tr: 68, epochs: 100, learning rate: 0.040; Accuracy (test, 99 instances): 71.7
#tr: 100, epochs: 100, learning rate: 0.040; Accuracy (test, 99 instances): 73.7
#tr: 140, epochs: 100, learning rate: 0.040; Accuracy (test, 99 instances): 75.8
#tr: 200, epochs: 100, learning rate: 0.040; Accuracy (test, 99 instances): 78.8
#tr: 239, epochs: 100, learning rate: 0.040; Accuracy (test, 99 instances): 78.8
#tr: 299, epochs: 100, learning rate: 0.040; Accuracy (test, 99 instances): 78.8
#tr: 347, epochs: 100, learning rate: 0.040; Accuracy (test, 99 instances): 79.8
#tr: 399, epochs: 100, learning rate: 0.040; Accuracy (test, 99 instances): 79.8
#tr: 20, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 65.7
#tr: 40, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 68.7
#tr: 68, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 71.7
#tr: 100, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 75.8
#tr: 140, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 74.7
#tr: 200, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 77.8
#tr: 239, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 78.8
#tr: 299, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 76.8
#tr: 347, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 79.8
#tr: 399, epochs: 100, learning rate: 0.050; Accuracy (test, 99 instances): 79.8
```

Double-click (or enter) to edit

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:

- Do you need to train with all the training dataset to get the highest accuracy with the test dataset?
- How do you justify that training the second run obtains worse accuracy than the first one (despite the second one uses more training data)?

```
# 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
```

- Can you get higher accuracy with additional hyperparameters (higher than `80.0`)?
- Is it always worth training for more epochs (while keeping all other hyperparameters fixed)?

TODO Add your answer here (code and text)

1. The results of the experiments with different hyperparameters show that the performance of the perceptron algorithm can be significantly affected by the choice of hyperparameters. The accuracy of the model is generally higher when trained with more training data and a higher learning rate. However, increasing the number of epochs does not always lead to improved performance.
2. In the first experiment with 100 training instances, 20 epochs, and a learning rate of 0.05, the test accuracy was 71%. In the second experiment with 200 training instances, 20 epochs, and a learning rate of 0.005, the test accuracy was lower at 68%. This suggests that increasing the number of training instances does not always lead to improved performance, especially when the learning rate is low.

Moreover, the second experiment's result shows that training with more data does not necessarily improve the accuracy, especially when the learning rate is low. In contrast, the first experiment's learning rate of 0.05 was relatively high, which might have helped to achieve higher accuracy, even with a smaller training set.

3. It is also worth noting that the highest accuracy achieved in the experiments was only 71%. This suggests that the perceptron algorithm may not be suitable for more complex datasets and that other machine learning models may be more appropriate.
4. Regarding the number of epochs, it is not always worth training for more epochs while keeping all other hyperparameters fixed. The third experiment with 100 training instances, 50 epochs, and a learning rate of 0.05 achieved a lower accuracy of 66% than the first experiment with only 20 epochs. This suggests that training for more epochs may lead to overfitting and a decrease in generalization performance.

Therefore, finding the optimal number of epochs that leads to the best performance on the validation set is essential for improving the model's performance.

In summary, the choice of hyperparameters, including the number of training instances, learning rate, and number of epochs, can significantly affect the performance of the perceptron algorithm. It is important to choose appropriate hyperparameters to achieve the best performance. Furthermore, increasing the number of training instances does not necessarily lead to improved accuracy, and training for more epochs does not always improve the model's performance.

✓ 51s completed at 9:12 PM