dy-bogasamudram-hw2-the-perceptron

February 18, 2023

1 CSCE 5218 / CSCE 4930 Deep Learning

2 HW1a The Perceptron (20 pt)

[3]:
```
# Get the datasets
!wget http://huang.eng.unt.edu/CSCE-5218/test.dat
!wget http://huang.eng.unt.edu/CSCE-5218/train.dat
```

```
--2023-02-18 03:31:54--  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

2023-02-18 03:31:54 (305 MB/s) - ‘test.dat’ saved [2844/2844]

--2023-02-18 03:31:54--  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

2023-02-18 03:31:54 (214 MB/s) - ‘train.dat’ saved [11244/11244]
```

[4]:
```
# 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
```

2.0.1 Build the Perceptron Model

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

[5]:
```python
import math
import itertools
import re
```

```python
# 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):
    #TODO: Return dot product of array 1 and array 2
    return sum([x * y for x, y in zip(array1, array2)])

def sigmoid(x):
    #TODO: Return outpout of sigmoid function on 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):
    #TODO: return the output of the model
    return sigmoid(dot_product(weight, instance))
# 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):
    #TODO: return the prediction of the model
    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
    #Initialize weights to 0
    weights = [0] * (len(instances[0])-1)

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

    return weights
```

[5]:

```
```

2.1 Run it

[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
```

2.2 Questions

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

2.2.1 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 predict function already calls the output function to compute the output value of the model on the instance, so we don't require the code snippet output= predict(weights, instance) in the train perceptron function. To determine the output value of the model, the output function computes the dot product of the weight vector and the instance vector and then runs the result through the sigmoid function. The dot product of the weight vector and the instance vector would therefore be computed twice if the predict function were to be called in the train perceptron function: once in the predict function and once in the output function that the predict function called.

2.2.2 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) #The code and output along with the accuracy and obtained below.

[7]:
```
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 with
num_epochs = [5, 10, 20, 50, 100]      # number of epochs
lr_array = [0.005, 0.01, 0.05]         # learning rate

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}")
```

```
#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
```

2.2.3 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: - A. Do you need to train with all the training dataset to get the highest accuracy with the test dataset? - 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)? `#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`)? - D. Is it always worth training for more epochs (while keeping all other hyperparameters fixed)?

TODO: Add your answer here (code and text) A)The outcomes with various hyperparameters demonstrate that improving performance on the test set does not always result from raising the percentage of the training dataset.

B) It is clear that improving performance did not always result from increasing the training data from 100 to 200 instances in the second run.

C) We may achieve greater accuracy with additional hyperparameters, but we were unable to identify any such factors in our testing.

D) It is clear from the output that training over a larger number of epochs does not always result in improved performance on the test set.