Started on	Saturday, 7 October 2023, 4:32 PM
State	Finished
Completed on	Saturday, 7 October 2023, 9:27 PM
Time taken	4 hours 55 mins
Marks	6.00/6.00
Grade	**1.00** out of 1.00 (**100**%)

Question 1

Correct

Mark 1.00 out of 1.00

The k-Nearest Neighbours (kNN) algorithm depends on two functions that must be passed to the algorithm:

- *distance*: this is a function that takes two objects and returns a non-negative number that is the distance between the objects according to some metric. This function is used to identify the neighbours of an object.
- *combine*: this is a function that takes a set of outputs and combines them in order to derive a new prediction.

In this question you have to write two concrete examples of these functions. Write the following functions:

- `euclidean_distance(v1, v2)` where v1 and v2 are two numeric vectors (non-empty sequences) with the same number of elements. The function must return the Euclidean distance between the points represented by v1 and *v2*.
- `majority_element(labels)` where `labels` is a non-empty collection of class labels. The function must return a label that has the highest frequency (most common). [if there is a tie it doesn't matter which majority is returned.] This is an example of a combine function.

For example:

Test	Result

`from student_answer import euclidean_distance`
 	
`print(euclidean_distance([0, 3, 1, -3, 4.5],[-2.1, 1, 8, 1, 1]))`
	9.25526876973327

`from student_answer import majority_element`
 	
`print(majority_element([0, 0, 0, 0, 0, 1, 1, 1]))`
`print(majority_element("ababc") in "ab")`
	0
	True

Answer: (penalty regime: 0, 15, ... %)

```
 1  def euclidean_distance(v1, v2):
 2      """return the Euclidean distance between the points represented by v1 and v
 3      square_distance = 0
 4      for i in range(len(v1)):
 5          square_distance += (v1[i] - v2[i]) ** 2
 6      return square_distance ** 0.5
 7
 8  def majority_element(labels):
 9      """labels is a non-empty collection of class labels. The function must
10      return a label that has the highest frequency (most common).
11      [if there is a tie it doesn't matter which majority is returned.]
12      This is an example of a combine function.
13      """
14      num_dict = {}
15      for element in labels:
16          num_dict[element] = num_dict.get(element, 0) + 1
17      most = 0
18      for key, value in num_dict.items():
19          if value > most:
20              most = value
21              ans = key
22
```

	Test	Expected	Got	

✔	`from student_answer import euclidean_distance`
	
	`print(euclidean_distance([0, 3, 1, -3, 4.5],[-2.1, 1, 8, 1, 1]))`
		9.25526876973327	9.25526876973327	✔

✔	`from student_answer import majority_element`
	
	`print(majority_element([0, 0, 0, 0, 0, 1, 1, 1]))`
	`print(majority_element("ababc") in "ab")`
		0
		True
			0
			True
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 2

Correct

Mark 1.00 out of 1.00

Write a function `knn_predict(input, examples, distance, combine, k)` that takes an input and predicts the output by combining the output of the *k* nearest neighbours. If after selecting *k* nearest neighbours, the distance to the farthest selected neighbour and the distance to the nearest unselected neighbour are the same, more neighbours must be selected until these two distances become different or all the examples are selected. The description of the parameters of the function are as the following:

- `input`: an input object whose output must be predicted. Do not make any assumption about the type of input other than that it can be consumed by the distance function.
- `examples`: a collection of pairs. In each pair the first element is of type input and the second element is of type output.
- `distance`: a function that takes two objects and returns a non-negative number that is the distance between the two objects according to some metric.
- `combine`: a function that takes a set of outputs and combines them in order to derive a new prediction (output).
- `k`: a positive integer which is the number of nearest neighbours to be selected. If there is a tie more neighbours will be selected (see the description above).

Notes

- You only need to provide a single function `knn_predict` and the related import statements. You do not need to provide any *distance* or *combine* functions. These functions are defined in the test cases and are passed to your function as arguments.
- The `majority_element` function used in some test cases returns the smallest element when there is a tie. For example `majority_element('--++')` returns `'+'` because it is the most common label (like `-`) and in the character encoding system `'+'` comes before `'-'`.

For example:

Test	Result
``from student_answer import knn_predict

examples = [
    ([2], '-'),
    ([3], '-'),
    ([5], '+'),
    ([8], '+'),
    ([9], '+'),
]

distance = euclidean_distance
combine = majority_element

for k in range(1, 6, 2):
    print("k =", k)
    print("x", "prediction")
    for x in range(0,10):
        print(x, knn_predict([x], examples, distance, combine, k))
    print()
``` | ```
k = 1
x prediction
0 -
1 -
2 -
3 -
4 +
5 +
6 +
7 +
8 +
9 +

k = 3
x prediction
0 -
1 -
2 -
3 -
4 -
5 +
6 +
7 +
8 +
9 +

k = 5
x prediction
0 +
1 +
2 +
3 +
4 +
5 +
6 +
7 +
8 +
9 +
``` |

Test	Result

``python
from student_answer import knn_predict
 	
# using knn for predicting numeric values 	
 	
examples = [
    ([1], 5),
    ([2], -1),
    ([5], 1),
    ([7], 4),
    ([9], 8),
]
 	
def average(values):
    return sum(values) / len(values)
 	
distance = euclidean_distance
combine = average
 	
for k in range(1, 6, 2):
    print("k =", k)
    print("x", "prediction")
    for x in range(0,10):
        print("{} {:4.2f}".format(x, knn_predict([x], examples, distance, combine, k)))
    print()
``
	k = 1
	x prediction
	0 5.00
	1 5.00
	2 -1.00
	3 -1.00
	4 1.00
	5 1.00
	6 2.50
	7 4.00
	8 6.00
	9 8.00
	
	k = 3
	x prediction
	0 1.67
	1 1.67
	2 1.67
	3 1.67
	4 2.25
	5 1.33
	6 4.33
	7 4.33
	8 4.33
	9 4.33
	
	k = 5
	x prediction
	0 3.40
	1 3.40
	2 3.40
	3 3.40
	4 3.40
	5 3.40
	6 3.40
	7 3.40
	8 3.40
	9 3.40

Answer: (penalty regime: 0, 15, … %)

```python
def knn_predict(input, examples, distance, combine, k):
    """takes an input and predicts the output by combining the output of the
    k nearest neighbours. If after selecting k nearest neighbours, the distance
    to the farthest selected neighbour and the distance to the nearest
    unselected neighbour are the same, more neighbours must be selected
    until these two distances become different or all the examples are selected.
    """
    labels = [] #contain k or more outputs and transfer it to combine
    distances = [] #length is same as that of examples, store (distance, index)
    for i in range(len(examples)):
        distances.append((distance(input, examples[i][0]), i))
    distances.sort()
    last_value = None
    for value, index in distances:
        if k > 0:
            k -= 1
            labels.append(examples[index][1])
        elif last_value == value:
            labels.append(examples[index][1])
        else:
            break
        last_value = value
    return combine(labels)
```

	Test	Expected	Got	
| ✔ | ```python
from student_answer import knn_predict

examples = [
    ([2], '-'),
    ([3], '-'),
    ([5], '+'),
    ([8], '+'),
    ([9], '+'),
]

distance = euclidean_distance
combine = majority_element

for k in range(1, 6, 2):
    print("k =", k)
    print("x", "prediction")
    for x in range(0,10):
        print(x, knn_predict([x], examples, distance, combine, k))
    print()
``` | k = 1<br>x prediction<br>0 -<br>1 -<br>2 -<br>3 -<br>4 +<br>5 +<br>6 +<br>7 +<br>8 +<br>9 +<br><br>k = 3<br>x prediction<br>0 -<br>1 -<br>2 -<br>3 -<br>4 -<br>5 +<br>6 +<br>7 +<br>8 +<br>9 +<br><br>k = 5<br>x prediction<br>0 +<br>1 +<br>2 +<br>3 +<br>4 +<br>5 +<br>6 +<br>7 +<br>8 +<br>9 + | k = 1<br>x prediction<br>0 -<br>1 -<br>2 -<br>3 -<br>4 +<br>5 +<br>6 +<br>7 +<br>8 +<br>9 +<br><br>k = 3<br>x prediction<br>0 -<br>1 -<br>2 -<br>3 -<br>4 -<br>5 +<br>6 +<br>7 +<br>8 +<br>9 +<br><br>k = 5<br>x prediction<br>0 +<br>1 +<br>2 +<br>3 +<br>4 +<br>5 +<br>6 +<br>7 +<br>8 +<br>9 + | ✔ |

	Test	Expected	Got	
✔	```from student_answer import knn_predict

using knn for predicting numeric values

examples = [
    ([1], 5),
    ([2], -1),
    ([5], 1),
    ([7], 4),
    ([9], 8),
]

def average(values):
    return sum(values) / len(values)

distance = euclidean_distance
combine = average

for k in range(1, 6, 2):
    print("k =", k)
    print("x", "prediction")
    for x in range(0,10):
        print("{} {:4.2f}".format(x, knn_predict([x], examples, distance, combine, k)))
    print()
``` | k = 1
x prediction
0 5.00
1 5.00
2 -1.00
3 -1.00
4 1.00
5 1.00
6 2.50
7 4.00
8 6.00
9 8.00

k = 3
x prediction
0 1.67
1 1.67
2 1.67
3 1.67
4 2.25
5 1.33
6 4.33
7 4.33
8 4.33
9 4.33

k = 5
x prediction
0 3.40
1 3.40
2 3.40
3 3.40
4 3.40
5 3.40
6 3.40
7 3.40
8 3.40
9 3.40 | k = 1
x prediction
0 5.00
1 5.00
2 -1.00
3 -1.00
4 1.00
5 1.00
6 2.50
7 4.00
8 6.00
9 8.00

k = 3
x prediction
0 1.67
1 1.67
2 1.67
3 1.67
4 2.25
5 1.33
6 4.33
7 4.33
8 4.33
9 4.33

k = 5
x prediction
0 3.40
1 3.40
2 3.40
3 3.40
4 3.40
5 3.40
6 3.40
7 3.40
8 3.40
9 3.40 | ✔ |

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 3

Correct

Mark 1.00 out of 1.00

A perceptron is a function that takes a vector (list of numbers) of size *n* and returns 0 or 1 according to the definition of perceptron.

Write a function `construct_perceptron(weights, bias)` where `weights` is a vector (list of numbers) of of length *n* and `bias` is a scalar number and returns the corresponding perceptron function.

For example:

Test	Result

`from student_answer import construct_perceptron`
 	
`weights = [2, -4]`
`bias = 0`
`perceptron = construct_perceptron(weights, bias)`
 	
`print(perceptron([1, 1]))`
`print(perceptron([2, 1]))`
`print(perceptron([3, 1]))`
`print(perceptron([-1, -1]))`
	0
	1
	1
	1

Answer: (penalty regime: 0, 15, ... %)

Reset answer

```
 1  def construct_perceptron(weights, bias):
 2      """Returns a perceptron function using the given paramers."""
 3      def perceptron(input):
 4          # Complete (a line or two)
 5
 6          # Note: we are masking the built-in input function but that is
 7          # fine since this only happens in the scope of this function and the
 8          # built-in input is not needed here.
 9          return int(sum([weights[i]*input[i] for i in range(len(input))]) + bias >
10
11      return perceptron # this line is fine
12
```

	Test	Expected	Got	

✔	`from student_answer import construct_perceptron`
	
	`weights = [2, -4]`
	`bias = 0`
	`perceptron = construct_perceptron(weights, bias)`
	
	`print(perceptron([1, 1]))`
	`print(perceptron([2, 1]))`
	`print(perceptron([3, 1]))`
	`print(perceptron([-1, -1]))`
		0
		1
		1
		1
			0
			1
			1
			1
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 4

Correct

Mark 1.00 out of 1.00

Write a function `accuracy(classifier, inputs, expected_outputs)` that passes each input in the sequence of inputs to the given classifier function (e.g. a perceptron) and compares the predictions with the expected outputs. The function must return the accuracy of the classifier on the given data. Accuracy must be a number between 0 and 1 (inclusive).

Note: an important application of a metric such as accuracy is to see how a classifier (e.g. a spam filter) performs on unseen data. In this case, the inputs must be some data that it has not seen during training but has been labeled by humans.

For example:

Test	Result

`from student_answer import accuracy`
 	
`perceptron = construct_perceptron([-1, 3], 2)`
`inputs = [[1, -1], [2, 1], [3, 1], [-1, -1]]`
`targets = [0, 1, 1, 0]`
 	
`print(accuracy(perceptron, inputs, targets))`
	0.75

Answer: (penalty regime: 0, 15, ... %)

```python
 1  def accuracy(classifier, inputs, expected_outputs):
 2      """passes each input in the sequence of inputs to the given classifier
 3      function (e.g. a perceptron) and compares the predictions with the
 4      expected outputs. The function must return the accuracy of the classifier
 5      on the given data. Accuracy must be a number between 0 and 1 (inclusive).
 6      """
 7      correct = 0
 8      for i in range(len(inputs)):
 9          if classifier(inputs[i]) == expected_outputs[i]:
10              correct += 1
11      return correct / len(inputs)
```

	Test	Expected	Got	

✔	`from student_answer import accuracy`
	
	`perceptron = construct_perceptron([-1, 3], 2)`
	`inputs = [[1, -1], [2, 1], [3, 1], [-1, -1]]`
	`targets = [0, 1, 1, 0]`
	
	`print(accuracy(perceptron, inputs, targets))`
		0.75	0.75	✔

✔	`from student_answer import accuracy`
	
	`perceptron = construct_perceptron([1, -3], 2)`
	`inputs = [[1, -1], [2, 1], [3, 1], [-1, -1]]`
	`targets = [0, 1, 1, 0]`
	
	`print(accuracy(perceptron, inputs, targets))`
		0.5	0.5	✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question 5

Correct

Mark 1.00 out of 1.00

Consider a binary classification problem (i.e. there are two classes in the domain) where each object is represented by 2 numeric values (2 features). We are using a single perceptron as a classifier for this domain and want to learn its parameters. The weight update rule is $w \leftarrow w + \eta x(t - y)$. We use the following configuration.

```
weights = [-0.5, 0.5]
bias = -0.5
learning_rate = 0.5

examples = [
    ([1, 1],   0),    # index 0 (first example)
    ([2, 0],   1),
    ([1, -1],  0),
    ([-1, -1], 1),
    ([-2, 0],  0),
    ([-1, 1],  1),
]
```

Answer the following with numeric values. Do <u>not</u> use fractions.

- After seeing the example at index 0, the value of the weight vector is [-0.5 ✔ , 0.5 ✔] and the value of bias is
  -0.5 ✔ .

- After seeing the example at index 1, the value of the weight vector is [0.5 ✔ , 0.5 ✔] and the value of bias is
  0 ✔ .

- After seeing the example at index 2, the value of the weight vector is [0 ✔ , 1 ✔] and the value of bias is
  -0.5 ✔ .

Correct

Marks for this submission: 1.00/1.00.

Question 6

Correct

Mark 1.00 out of 1.00

Write a function `learn_perceptron_parameters(weights, bias, training_examples, learning_rate, max_epochs)` that adjusts the weights and bias by iterating through the training data and applying the perceptron learning rule. The function must return a pair (2-tuple) where the first element is the vector (list) of adjusted weights and second argument is the adjusted bias. The parameters of the function are:

- `weights`: an array (list) of initial weights of length n
- `bias`: a scalar number which is the initial bias
- `training_examples`: a list of training examples where each example is a pair. The first element of the pair is a vector (tuple) of length n. The second element of the pair is an integer which is either 0 or 1 representing the negative or positive class correspondingly.
- `learning_rate`: a positive number representing eta in the learning equations of perceptron.
- `max_epochs`: the maximum number of times the learner is allowed to iterate through all the training examples.

For example:

Test	Result
``from student_answer import learn_perceptron_parameters

weights = [2, -4]
bias = 0
learning_rate = 0.5
examples = [
  ((0, 0), 0),
  ((0, 1), 0),
  ((1, 0), 0),
  ((1, 1), 1),
  ]
max_epochs = 50

weights, bias = learn_perceptron_parameters(weights, bias, examples, learning_rate, max_epochs)
print(f"Weights: {weights}")
print(f"Bias: {bias}\n")

perceptron = construct_perceptron(weights, bias)

print(perceptron((0,0)))
print(perceptron((0,1)))
print(perceptron((1,0)))
print(perceptron((1,1)))
print(perceptron((2,2)))
print(perceptron((-3,-3)))
print(perceptron((3,-1)))
``` | ```Weights: [1.0, 0.5]
Bias: -1.5

0
0
0
1
1
0
1
``` |
| ```from student_answer import learn_perceptron_parameters

weights = [2, -4]
bias = 0
learning_rate = 0.5
examples = [
  ((0, 0), 0),
  ((0, 1), 1),
  ((1, 0), 1),
  ((1, 1), 0),
  ]
max_epochs = 50

weights, bias = learn_perceptron_parameters(weights, bias, examples, learning_rate, max_epochs)
print(f"Weights: {weights}")
print(f"Bias: {bias}\n")
``` | ```Weights: [-0.5, -0.5]
Bias: 0.0
``` |

Answer: (penalty regime: 0, 15, ... %)

```
1 ▾ def learn_perceptron_parameters(weights, bias, training_examples, learning_rate, ▮
2        """adjusts the weights and bias by iterating through the training data and
3        applying the perceptron learning rule. The function must
4        return a pair (2-tuple) where the first element is the vector (list)
5        of adjusted weights and second argument is the adjusted bias.
6        """
7        epoch = 1
```

```
  7        epoch = 1
  8        finished = False
  9        weights = weights[:]
 10 ▼      while not finished and epoch <= max_epochs:
 11            epoch += 1
 12            finished = True
 13 ▼          for example, expected in training_examples:
 14                prediction = int(sum([weights[i]*example[i] for i in range(len(exampl
 15 ▼              if prediction != expected:
 16                    finished = False
 17                    bias += learning_rate * (expected - prediction)
 18                    weights = [weights[i] + learning_rate*example[i]*(expected-predic
 19        return weights, bias
```

	Test	Expected	Got	

✔	`from student_answer import learn_perceptron_parameters`
	
	`weights = [2, -4]`
	`bias = 0`
	`learning_rate = 0.5`
	`examples = [`
	`  ((0, 0), 0),`
	`  ((0, 1), 0),`
	`  ((1, 0), 0),`
	`  ((1, 1), 1),`
	`  ]`
	`max_epochs = 50`
	
	`weights, bias = learn_perceptron_parameters(weights, bias, examples,`
	`learning_rate, max_epochs)`
	`print(f"Weights: {weights}")`
	`print(f"Bias: {bias}\n")`
	
	`perceptron = construct_perceptron(weights, bias)`
	
	`print(perceptron((0,0)))`
	`print(perceptron((0,1)))`
	`print(perceptron((1,0)))`
	`print(perceptron((1,1)))`
	`print(perceptron((2,2)))`
	`print(perceptron((-3,-3)))`
	`print(perceptron((3,-1)))`
		Weights: [1.0, 0.5]
		Bias: -1.5
		
		0
		0
		0
		1
		1
		0
		1
			Weights: [1.0, 0.5]
			Bias: -1.5
			
			0
			0
			0
			1
			1
			0
			1
				✔

✔	`from student_answer import learn_perceptron_parameters`
	
	`weights = [2, -4]`
	`bias = 0`
	`learning_rate = 0.5`
	`examples = [`
	`  ((0, 0), 0),`
	`  ((0, 1), 1),`
	`  ((1, 0), 1),`
	`  ((1, 1), 0),`
	`  ]`
	`max_epochs = 50`
	
	`weights, bias = learn_perceptron_parameters(weights, bias, examples,`
	`learning_rate, max_epochs)`
	`print(f"Weights: {weights}")`
	`print(f"Bias: {bias}\n")`
		Weights: [-0.5, -0.5]
		Bias: 0.0
			Weights: [-0.5, -0.5]
			Bias: 0.0
				✔

Passed all tests! ✔

(Correct)

Marks for this submission: 1.00/1.00.