

This homework will aid in your understanding of the Perceptron Algorithm by having you:

- practice applying the perceptron algorithm to toy data
- implement the perceptron algorithm and one of its variants
- apply your perceptron implementation on larger, more interesting data sets

In addition to the Numpy functions and features mentioned in hw 1, here are some you should be familiar with for this assignment:

- `np.argmax`
- `np.array_split`, look at the `axis` argument
- `np.concatenate`, look at the `axis` argument
- `np.zeros`
- `numpy.ndarray.shape`
- numpy [logic functions](#) e.g. `np.array([[1, 2], [3, 4]]) == 3`

For this homework, it will be helpful to review the [notes](#) on perceptron.

For problems 7-10, [here](#) is a code file that will be useful for debugging on your computer; alternatively, you may find it helpful to use [this preformatted google colab notebook](#).

**Note: for all of the problems in this assignment, you are allowed to use for loops.**

## 1) Perceptron Mistakes

Let's apply [the perceptron algorithm \(through the origin\)](#) to a small training set containing three points:

$i$	Data Points $x^{(i)}$	Labels $y^{(i)}$
1	$[1, -1]$	1
2	$[0, 1]$	-1
3	$[-1.5, -1]$	1

Given that the algorithm starts with  $\theta^{(0)} = [0, 0]$ , **the first point that the algorithm sees is always a mistake**. The algorithm starts with **some** data point (to be specified in the question), and then cycles through the data until it makes no further mistakes.

### 1.1) Take 1

**1.1a)** How many mistakes does the algorithm make until convergence if the algorithm starts with data point  $x^{(1)}$ ?

Number of mistakes is:

Ask for Help

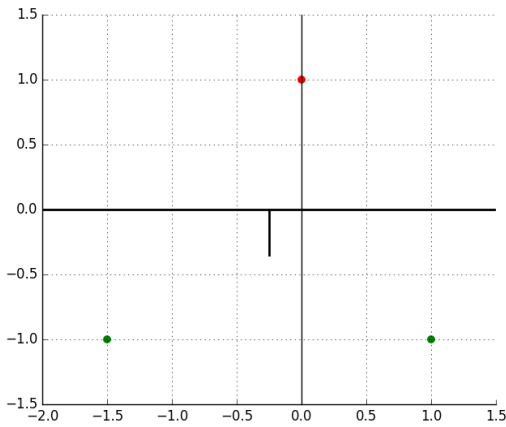
100.00%

You have infinitely many submissions remaining.

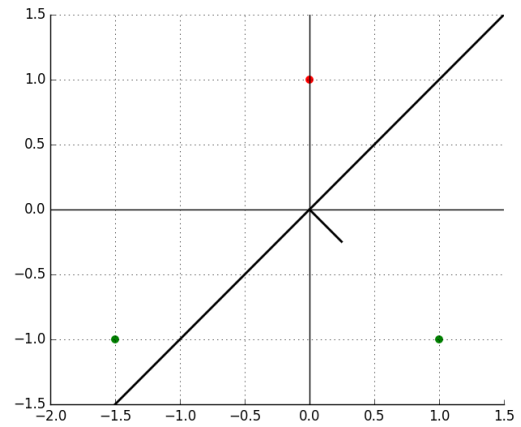
Solution: 2

### Explanation:

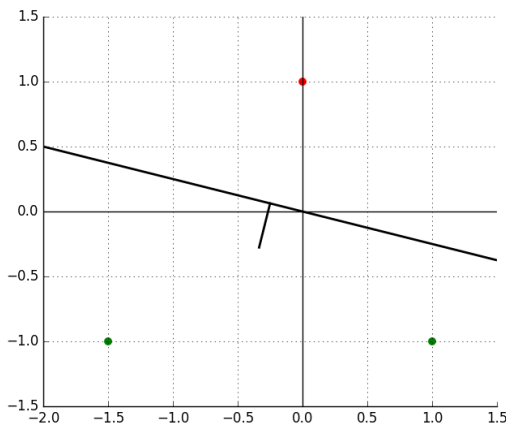
During each iteration of the perceptron, if the current classifier incorrectly classifies the current point, i.e. it does not predict the label  $y^{(i)}$  for the input  $x^{(i)}$ , we update  $\theta$  to be  $\theta + y^{(i)}x^{(i)}$ . We see that the classifier makes mistakes on the 1st and 3rd data points before classifying all three correctly.



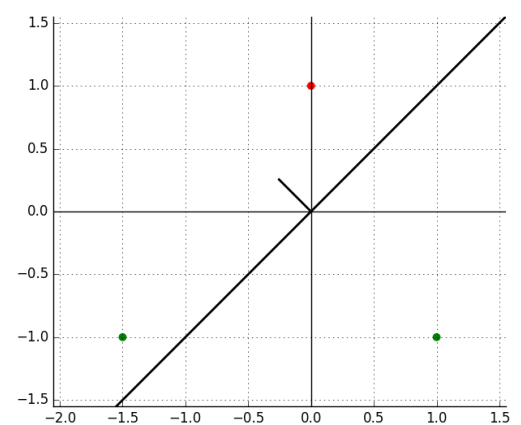
(1)



(2)



(3)



(4)

**1.1b)** Which of the above plot(s) correspond to the progression of the hyperplane as the algorithm cycles? Ignore the initial 0 weights, and include an entry only when  $\theta$  changes.

Please provide the plot number(s) in the order of progression as a Python list.

[Ask for Help](#)
**100.00%**

*You have infinitely many submissions remaining.*

Solution: [2, 3]

**Explanation:**  $\theta$  updates to [1,-1] and then [-.5,-2], corresponding to plots 2 and 3, respectively.

**1.1c)** How many mistakes does the algorithm make if it starts with data point  $x^{(2)}$  (and then does  $x^{(3)}$  and  $x^{(1)}$ )?

Number of mistakes is:

[Ask for Help](#)
**100.00%**

*You have infinitely many submissions remaining.*

Solution: 1

**Explanation:**

After seeing  $x^{(2)}$ ,  $\theta$  updates to [0,-1] and properly classifies all the points.

**1.1d)** Again, if it starts with data point  $x^{(2)}$  (and then does  $x^{(3)}$  and  $x^{(1)}$ ), which plot(s) correspond to the progression of the hyperplane as the algorithm cycles? Ignore the initial 0 weights.

Please provide the plot number(s) in the order of progression as a Python list.

[Ask for Help](#)
**100.00%**

*You have infinitely many submissions remaining.*

Solution: [1]

**Explanation:**

After seeing  $x^{(2)}$ ,  $\theta$  updates to [0,-1], which corresponds to plot 1.

## 1.2) Take 2

Now assume that  $x^{(3)} = [-10, -1]$ , with label 1.

**1.2a)** How many mistakes does the algorithm make until convergence if cycling starts with data point  $x^{(1)}$ ?

Number of mistakes is

Ask for Help

100.00%

*You have infinitely many submissions remaining.*

Solution: 6

**Explanation:**

Before classifying all three points correctly,  $\theta$  is updated as follows:  $[1, -1]$ ,  $[-9, -2]$ ,  $[-8, -3]$ ,  $[-7, -4]$ ,  $[-6, -5]$ ,  $[-5, -6]$  (making mistakes on points 1, 3, 1, 1, 1, 1, respectively).

**1.2b)** How many mistakes if it starts with data point  $x^{(2)}$ ?

Number of mistakes is

Ask for Help

100.00%

*You have infinitely many submissions remaining.*

Solution: 1

**Explanation:**  $\theta$  updates to  $[0, -1]$  and properly classifies all the points.

## 2) Initialization

**2.1)** If we were to initialize the perceptron algorithm with  $\theta = [1000, -1000]$ , how would it affect the number of mistakes made in order to separate the data set from question 1?

- ☐ It would have small or no effect on the number of mistakes.
- ☐ It would significantly decrease the number of mistakes.
- ☒ It would significantly increase the number of mistakes.

Ask for Help 100.00%

You have infinitely many submissions remaining.

Solution: It would significantly increase the number of mistakes.

#### Explanation:

When  $\theta$  is initialized with larger values, corrections to  $\theta$  have a smaller impact. Therefore, in general, it takes more corrections until  $\theta$  can separate the data points.

**2.2)** Provide a value of  $\theta^{(0)}$  for which running the perceptron algorithm (through origin) on the data set from question 1 returns a different result than using  $\theta^{(0)} = [0, 0]$ . The data set is repeated below:

$i$	Data Points $x^{(i)}$	Labels $y^{(i)}$
1	$[1, -1]$	1
2	$[0, 1]$	-1
3	$[-1.5, -1]$	1

Enter the 2 coordinates of  $\theta$  as a Python list or the string 'none' if none exists.'

Ask for Help 100.00%

You have infinitely many submissions remaining.

Solution: 'There are many possible solutions. One is  $[1, -2]$ .'

### 3) Dual View

The following table shows a data set and the number of times each point is misclassified during a run of the perceptron algorithm (with offset).  $\theta$  is initialized to the zero vector and  $\theta_0$  is initialized to 0.

$i$	$x^{(i)}$	$y^{(i)}$	times misclassified
1	$[-3, 2]$	1	2
2	$[-1, 1]$	-1	4
3	$[-1, -1]$	-1	2

$i$	$x^{(i)}$	$y^{(i)}$	times misclassified
4	[2, 2]	-1	1
5	[1, -1]	-1	0

3.1) What is the post training  $\theta$ ?

Provide it as a python list of the form  $[a, b]$ .

Ask for Help

100.00%

You have infinitely many submissions remaining.

Solution: [-2, 0]

#### Explanation:

Each time the  $i^{th}$  point is misclassified, we add  $y^{(i)}x^{(i)}$  to our current value of  $\theta$ . So, if  $n^{(i)}$  is the number of times the  $i^{th}$  point gets misclassified, our final value of  $\theta$  is  $\sum_{i=1}^5 n^{(i)}y^{(i)}x^{(i)} = 2 \cdot 1 \cdot [-3, 2] + 4 \cdot -1 \cdot [-1, 1] + 2 \cdot -1 \cdot [-1, -1] + 1 \cdot -1 \cdot [2, 2] + 0 \cdot -1 \cdot [1, -1] = [-2, 0]$ .

You can double-check this with the following code (once you have implemented `perceptron()`):

```
perceptron(np.array([-3,2],[-1,1],[-1,-1],[2,2],[1,-1]).T, np.array([[1,-1,-1,-1,-1]]))
```

3.2) What is the post training  $\theta_0$ ?

Provide it as a number.

Ask for Help

100.00%

You have infinitely many submissions remaining.

Solution: -5

#### Explanation:

Similar to the previous part, we add  $y^{(i)}$  each time the  $i^{th}$  point is misclassified. So, if  $n^{(i)}$  is the number of times the  $i^{th}$  point is misclassified,  $\theta_0 = \sum_{i=1}^5 n^{(i)}y^{(i)} = 2 \cdot 1 + 4 \cdot -1 + 2 \cdot -1 + 1 \cdot -1 + 0 \cdot -1 = -5$ .

## 4) Decision Boundaries

### 4.1) AND

Consider the AND function defined over three binary variables:  $f(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3)$ .

If you are unfamiliar with the AND function, it simply returns 1 if all three variables are true (value of 1) and 0 otherwise.

We aim to find a  $\theta$  such that, for any  $x = [x_1, x_2, x_3]$ , where  $x_i \in \{0, 1\}$ :

$$\theta \cdot x + \theta_0 > 0 \text{ when } f(x_1, x_2, x_3) = 1, \text{ and}$$

$$\theta \cdot x + \theta_0 \leq 0 \text{ when } f(x_1, x_2, x_3) = 0.$$

**4.1a)** For each of the combination of values of  $(x_1, x_2, x_3)$ , that is,  $[(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)]$  enter the values of  $f(x_1, x_2, x_3)$ .

Please enter the values of  $f(x_1, x_2, x_3)$  as a Python list.

[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Solution: `[0, 0, 0, 0, 0, 0, 0, 1]`

---

**Explanation:**

We simply AND each triple of values in the list.

**4.1b)** Assuming  $\theta_0 = 0$  (no offset), enter  $\theta$  as a Python list of length 3 or enter 'none' as a Python string (with quotes) if none exists.

Enter a Python list of 3 numbers or the string 'none'

[Ask for Help](#)

100.00%

*You have infinitely many submissions remaining.*

Solution: 'none'

### Explanation:

One way to visualize this is to picture the unit cube formed by the 8 values of  $x$ . The separator is a plane that must separate the far corner of the cube  $(1,1,1)$  from the other corners. Because the separator must pass through the origin, this is impossible.

Mathematically, we can prove this by contradiction. Assume that there does exist such a separator, and consider the four points  $x^{(2)} = (0, 0, 1)$ ,  $x^{(3)} = (0, 1, 0)$ ,  $x^{(5)} = (1, 0, 0)$ , and  $x^{(8)} = (1, 1, 1)$ .

$$0 \geq \theta \cdot (x^{(2)} + x^{(3)} + x^{(5)}) = \theta \cdot x^{(8)} > 0,$$

a contradiction.

**4.1c)** Assuming  $\theta_0$  is non-zero (offset), enter a  $\theta$  and  $\theta_0$  as a Python list of length 4 ( $\theta_0$  last) or enter 'none' as a Python string (with quotes) if none exists.



Enter a Python list with 4 numbers or the string 'none' .

[Ask for Help](#)

100.00%

You have infinitely many submissions remaining.

Solution: [1, 1, 1, -2.5]

### Explanation:

There are many combinations of values that work. Our solution observes that all three coordinates must be 1 for the point to receive a positive output, so our separator checks whether or not the sum of coordinates is above 2.5.

You can find this using the `perceptron()` function with this call:

```
perceptron(np.array([(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1)]).T,
            np.array([[0,0,0,0,0,0,0,1]])*2-1)
```

## 4.2) Families

You are given the following labeled data points:

- Positive examples:  $[-1, 1]$  and  $[1, -1]$ ,
- Negative examples:  $[1, 1]$  and  $[2, 2]$ .

For each of the following parameterized families of classifiers, find the parameters of a family member that can correctly classify the above data, or think about why no such family member exists.

Is there a classifier of the following forms that can correctly classify the above data?

Recall from lecture that we consider a point lying exactly on the separator to be classified as negative.

**4.2a)** Inside or outside of an origin-centered circle with radius  $r$

Enter a value for  $r$  or the string 'none' if none exists.'

[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Solution: 'none'

---

**Explanation:**

Positive points  $[-1, 1]$  and  $[1, -1]$  have the same distance from the origin as the negative point  $[1, 1]$ , so they cannot be separated by a circle at the origin.

**4.2b)** Inside or outside of a circle centered on  $x_0$  with radius  $r$

Enter a list with 3 entries for coordinates of  $x_0$  and  $r$  ( $[x_{0\_1}, x_{0\_2}, r]$ ) or the string 'none' if none exists.'

[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Solution:  $[1.5, 1.5, 1.0]$

---

**Explanation:**

There are many combinations of values that work. Our answer puts a circle centered halfway between the two negative points, with a radius small enough to exclude the positive points.

**4.2c)** On one side of a line through the origin with normal  $\theta$  (recall normal vector points into positive half-space).

Enter a list with 2 entries for coordinates of  $\theta$  or the string 'none' if none exists.'

[Ask for Help](#)
**100.00%**

*You have infinitely many submissions remaining.*

Solution: 'none '

#### Explanation:

Since positive points  $[-1,1]$  and  $[1,-1]$  are diametrically opposite the origin, no separator through the origin can classify both as positive.

**4.2d)** On one side of a line with normal  $\theta$  and offset  $\theta_0$  (recall normal vector points into positive half-space).

Enter a list with 3 entries for coordinates of  $\theta$  and  $\theta_0$  ( $[\theta_1, \theta_2, \theta_0]$ ) or the string 'none' if none exists.'

[Ask for Help](#)
**100.00%**

*You have infinitely many submissions remaining.*

Solution:  $[-1, -1, 1]$

#### Explanation:

There are many combinations of values that work. For example, we see that the line passing through  $[0,1]$  and  $[1,0]$  appropriately separates the points.  $\theta = [-1, -1]$  is normal to this line, and solving for  $\theta_0$ , we find  $\theta_0 = 1$ .

**4.2e)**

Which of the above are families of linear classifiers?

- ☐ 4.2a
- ☐ 4.2b
- ☒ 4.2c
- ☒ 4.2d

Ask for Help

100.00%

You have infinitely many submissions remaining.

Solution:

✗ 4.2a

✗ 4.2b

✓ 4.2c

✓ 4.2d

**Explanation:**

c and d both exactly follow the form of linear classifiers that we have studied. a and b use circles to separate points, requiring terms that are quadratic in  $x$ , e.g.  $x_1^2 + x_2^2 = r^2$ . They cannot be formed by linear combinations of the terms in  $x$ , and thus are not families of linear classifiers.

## 5) Separation

Indicate if the following datasets are:

- not linearly separable
- linearly separable without an offset
- linearly separable only with a non-zero offset

**HINT: You shouldn't have to work through the perceptron algorithm to figure this out.**

5.1) Dataset 1:

$i$	Data Points $x^{(i)}$	Labels $y^{(i)}$
1	$[1, -1]$	-1
2	$[1, 1]$	1
3	$[2, -1]$	1

$i$  Data Points  $x^{(i)}$  Labels  $y^{(i)}$

4 [2, 1] -1

This dataset is: Not linearly separable

Ask for Help

100.00%

You have infinitely many submissions remaining.

Solution: Not linearly separable

#### Explanation:

Note that the two points with positive labels are diagonally opposite corners of a square. A single line cannot be drawn such that both of these points lie on the same side.

### 5.2) Dataset 2:

$i$  Data Points  $x^{(i)}$  Labels  $y^{(i)}$

1 [1, -1] 1

2 [1, 1] 1

3 [2, -1] -1

4 [2, 1] -1

This dataset is: Linearly separable only with a non-zero offset

Ask for Help

100.00%

You have infinitely many submissions remaining.

Solution: Linearly separable only with a non-zero offset

#### Explanation:

Here we can imagine two vertically adjacent corners of a square sharing the same label. The line  $x = 1.5$  is one such separator. Note that no separator through the origin exists.

### 5.3) Dataset 3:

$i$  Data Points  $x^{(i)}$  Labels  $y^{(i)}$

1 [1, -1, 1] 1

2 [1, 1, 1] 1

3 [2, -1, 1] -1

4 [2, 1, 1] -1

This dataset is: Linearly separable without an offset ▼

Ask for Help

100.00%

*You have infinitely many submissions remaining.*

Solution: Linearly separable without an offset

---

**Explanation:**

Now we have an added dimension, effectively raising the square above the origin. Draw a picture to understand why there now exists a plane (note the separator is no longer a line) which can separate the points.

**5.4)** Compare datasets 2 and 3, and see if you can generalize what you learned from those two examples. Which of the following transformations allow one to transform *any* dataset that is only linearly separable *with* an offset to a dataset that is linearly separable *without* an offset, such that the transformation doesn't depend on the values of the datapoints? Select all which are valid:

This dataset is:







- ☐ Remove the final dimension of each of the datapoints
- ☐ Add an extra dimension to all of the datapoints using any (nonzero) numbers
- ☒ Add an extra dimension to all of the datapoints using the same (nonzero) number for each point
- ☒ Add an extra dimension to all of the datapoints using a 1 for each point
- ☐ Add an extra dimension to all of the data points, using 0 for each point
- ☐ A transformation with these properties is impossible

Ask for Help

100.00%

*You have infinitely many submissions remaining.*

Solution:

-  Remove the final dimension of each of the datapoints
-  Add an extra dimension to all of the datapoints using any (nonzero) numbers
-  Add an extra dimension to all of the datapoints using the same (nonzero) number for each point
-  Add an extra dimension to all of the datapoints using a 1 for each point
-  Add an extra dimension to all of the data points, using 0 for each point
-  A transformation with these properties is impossible

---

**Explanation:**

Adding a dimension to each point with the same value for each point will work. Mathematically, a dot product in this higher dimensional space will be equivalent to a dot product in the lower dimensional space with the addition of a bias term. Due to this, many machine learning texts assume a 1 is appended to each point instead of explicitly writing a bias term. In this case the final element of the theta vector will be equal to  $\theta_0$  in the lower dimensional case.

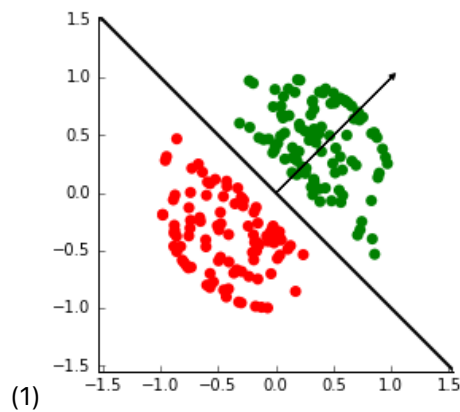
## 6) Mistakes and generalization

Please refer to [the Perceptron Convergence Theorem](#).

### 6.1) Plots

Consider the following plots. For each one estimate plausible values of  $R$  (an upper bound on the magnitude of the training vectors) and  $\gamma$  (the margin of the separator for the dataset). Consider values of  $R$  in the range  $[1, 10]$  and values of  $\gamma$  in the range  $[0.01, 2]$ .

6.1a)



Enter a Python list with 2 floats, a value of  $R$  and a value of  $\gamma$ :

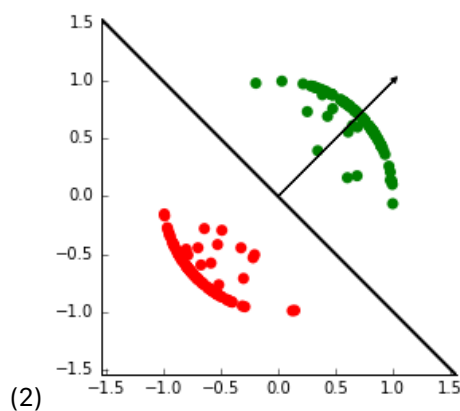
[Ask for Help](#)

100.00%

You have infinitely many submissions remaining.

Solution: `[1, 0.2]`

6.1b)





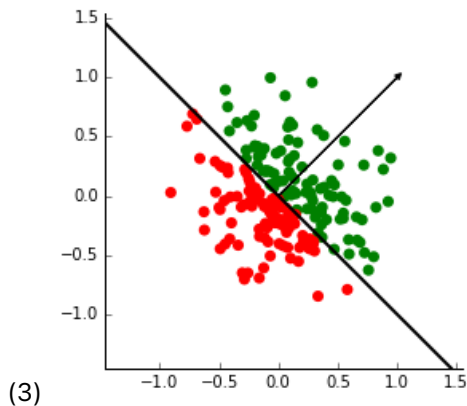
Enter a Python list with 2 floats, a value of R and a value of gamma:

[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Solution: [1, 0.5]

6.1c)



Enter a Python list with 2 floats, a value of R and a value of gamma:

[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

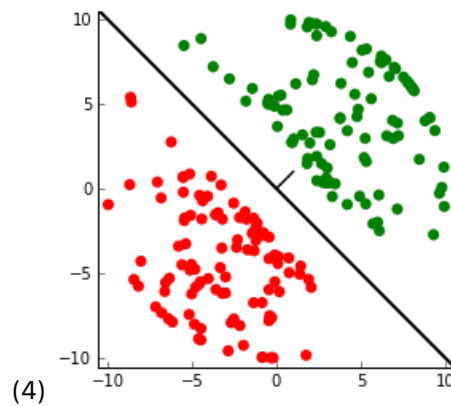
Solution: [1, 0.01]

---

**Explanation:**

The margin is clearly very small, because all the points go right up to the separator. In fact, one red dot does look like it may be on the wrong side of the separator, so the margin may even be negative.

6.1d)



Enter a Python list with 2 floats, a value of  $R$  and a value of  $\gamma$ :

Ask for Help

100.00%

You have infinitely many submissions remaining.

Solution: [10, 2]

## 6.2) Mistake Bound

**6.2a)** What is an upper bound on mistakes when  $R = 1$ ,  $n = 1000$  for each of the following values of  $\gamma$ ?

(.00001, .0001, .001, .01, .1, .2, .5)

Recall that given a linear separator that passes through the origin, the perceptron algorithm makes at most  $(\frac{R}{\gamma})^2$  mistakes.

**Check yourself:** Think about how you can compute this bound when the separator does not pass through the origin.

Enter a Python list with 7 floats.

Ask for Help

100.00%

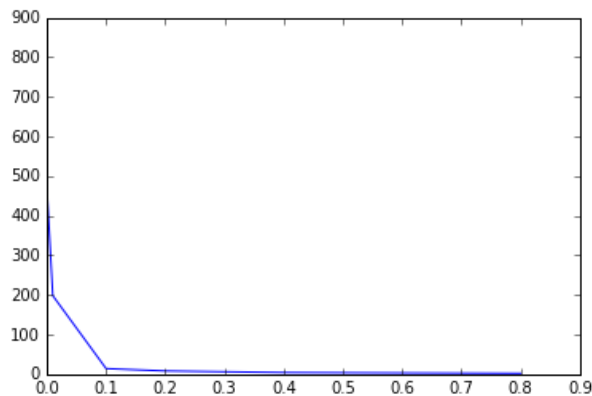
You have infinitely many submissions remaining.

Solution: [10000000000.0, 100000000.0, 1000000.0, 10000.0, 100, 25, 4]

As a sanity check, we have provided a plot for  $R = 1$ ,  $d = 4$ ,  $n = 1000$  of the actual numbers of mistakes made by the perceptron on one particular run, as a function of  $\gamma$ .

**Check yourself:** Make sure you understand this plot and why it agrees with our theoretical bounds.

The actual numbers of mistakes (on y axis) are: [862, 414, 446, 198, 14, 8, 4].



## 7) Implement perceptron

For this problem and the remaining problems below, [here](#) is a code file that will be useful for debugging on your computer; alternatively, you may find it helpful to use [this preformatted google colab notebook](#).

7) Implement [the perceptron algorithm](#), where

- `data` is a numpy array of dimension  $d$  by  $n$
- `labels` is numpy array of dimension 1 by  $n$
- `params` is a dictionary specifying extra parameters to this algorithm; your algorithm should run a number of iterations equal to  $T$
- `hook` is either `None` or a function that takes the tuple  $(\theta, \theta_0)$  as an argument and displays the separator graphically. We won't be testing this in the Tutor, but it will help you in debugging on your own machine.

It should return a tuple of  $\theta$  (a  $d$  by 1 array) and  $\theta_0$  (a 1 by 1 array).

We have given you some data sets in the code file for you to test your implementation.

Your function should initialize all parameters to 0, then run through the data, in the order it is given, performing an update to the parameters whenever the current parameters would make a mistake on that data point. Perform  $T$  iterations through the data. After every parameter update, if `hook` is defined, call it on the current  $(\theta, \theta_0)$  (as a single parameter in a python tuple).

When debugging on your own, you can use the procedure `test_linear_classifier` for testing. By default, it pauses after every parameter update to show the separator. For data sets not in 2D, or just to get the answer, set `draw = False`.

**See the top of problem 7 for the the code distribution.**

```

1 import numpy as np
2
3 def perceptron(data, labels, params={}, hook=None):
4     # if T not in params, default to 100
5     T = params.get('T', 100)
6
7     # if T not in params, default to 100
8     T = params.get('T', 100)
9     # Your implementation here
10    n = data.shape[1]
11    d = data.shape[0]
12    th = np.zeros(d)
13    th0 = np.array(0.0)

```

[Ask for Help](#)

100.00%

You have infinitely many submissions remaining.

Here is the solution we wrote:

```

import numpy as np

# x is dimension d by 1
# th is dimension d by 1
# th0 is dimension 1 by 1
# return 1 by 1 matrix of +1, 0, -1
def positive(x, th, th0):
    return np.sign(th.T@x + th0)

# Perceptron algorithm with offset.
# data is dimension d by n
# labels is dimension 1 by n
# T is a positive integer number of steps to run
# Perceptron algorithm with offset.
# data is dimension d by n
# labels is dimension 1 by n
# T is a positive integer number of steps to run
def perceptron(data, labels, params = {}, hook = None):
    # if T not in params, default to 100
    T = params.get('T', 100)
    (d, n) = data.shape

    theta = np.zeros((d, 1)); theta_0 = np.zeros((1, 1))
    for t in range(T):
        for i in range(n):
            x = data[:,i:i+1]
            y = labels[:,i:i+1]
            if y * positive(x, theta, theta_0) <= 0.0:
                theta = theta + y * x
                theta_0 = theta_0 + y
            if hook: hook((theta, theta_0))
    return theta, theta_0
#Note, the solution doesn't have to pretty; it's far better that it is understandable.

```

## 8) Implement averaged perceptron

Regular perceptron can be somewhat sensitive to the most recent examples that it sees. Instead, averaged perceptron produces a more stable output by outputting the average value of  $\mathbf{th}$  and  $\mathbf{th}_0$  across all iterations.

Implement averaged perceptron with the same spec as regular perceptron, and using the pseudocode below as a guide.

```

procedure averaged_perceptron({(x_(i), y_(i)), i=1,...,n}, T)
  th = 0 (d by 1); th0 = 0 (1 by 1)
  ths = 0 (d by 1); th0s = 0 (1 by 1)
  for t = 1,...,T do:
    for i = 1,...,n do:
      if y_(i)(th . x_(i) + th0) <= 0 then
        th = th + y_(i)x_(i)
        th0 = th0 + y_(i)
      ths = ths + th
      th0s = th0s + th0
  return ths/(nT), th0s/(nT)

```

```

1 import numpy as np
2
3 def averaged_perceptron(data, labels, params={}, hook=None):
4     # if T not in params, default to 100
5     T = params.get('T', 100)
6
7     # Your implementation here
8     n = data.shape[1]
9     d = data.shape[0]
10
11     th = np.zeros(d)
12     th0 = np.array(0)
13     ths = np.zeros(d)

```

[Ask for Help](#)

100.00%

You have infinitely many submissions remaining.

Here is the solution we wrote:

```

import numpy as np

# x is dimension d by 1
# th is dimension d by 1
# th0 is dimension 1 by 1
# return 1 by 1 matrix of +1, 0, -1
def positive(x, th, th0):
    return np.sign(th.T @ x + th0)

def averaged_perceptron(data, labels, params = {}, hook = None):
    T = params.get('T', 100)
    (d, n) = data.shape

    theta = np.zeros((d, 1)); theta_0 = np.zeros((1, 1))
    theta_sum = theta.copy()

    theta_0_sum = theta_0.copy()
    for t in range(T):
        for i in range(n):
            x = data[:, i:i+1]
            y = labels[:, i:i+1]
            if y * positive(x, theta, theta_0) <= 0.0:
                theta = theta + y * x
                theta_0 = theta_0 + y
                if hook: hook((theta, theta_0))
            theta_sum = theta_sum + theta
            theta_0_sum = theta_0_sum + theta_0
    theta_avg = theta_sum / (T*n)
    theta_0_avg = theta_0_sum / (T*n)
    if hook: hook((theta_avg, theta_0_avg))
    return theta_avg, theta_0_avg

```

## 9) Implement evaluation strategies

## 9.1) Evaluating a classifier

To evaluate a classifier, we are interested in how well it performs on data that it wasn't trained on. Construct a testing procedure that uses a training data set, calls a learning algorithm to get a linear separator (a tuple of  $\theta$ ,  $\theta_0$ ), and then reports the percentage correct on a new testing set as a float between 0. and 1..

The learning algorithm is passed as a function that takes a data array and a labels vector. Your evaluator should be able to interchangeably evaluate `perceptron` or `averaged_perceptron` (or future algorithms with the same spec), depending on what is passed through the `learner` parameter.

Assume that you have available the function `score` from HW 1, which takes inputs:

- `data`: a `d` by `n` array of floats (representing `n` data points in `d` dimensions)
- `labels`: a 1 by `n` array of elements in `(+1, -1)`, representing target labels
- `th`: a `d` by 1 array of floats that together with
- `th0`: a single scalar or 1 by 1 array, represents a hyperplane

and returns a scalar number of data points that the separator correctly classified.

The `eval_classifier` function should accept the following parameters:

- `learner` - a function, such as `perceptron` or `averaged_perceptron`
- `data_train` - training data
- `labels_train` - training labels
- `data_test` - test data
- `labels_test` - test labels

and returns the percentage correct on a new testing set as a float between 0. and 1..

```

1 import numpy as np
2
3 def eval_classifier(learner, data_train, labels_train, data_test, labels_test):
4     th, th0 = learner(data_train, labels_train)
5
6     train_results = th.T@data_train + th0
7     results_map = np.sign(train_results) == labels_test
8
9     return np.sum(results_map)/results_map.shape[1]
10

```

[Ask for Help](#)

50.00%

You have infinitely many submissions remaining.

Here is the solution we wrote:

```

import numpy as np
def eval_classifier(learner, data_train, labels_train, data_test, labels_test):
    th, th0 = learner(data_train, labels_train)
    return score(data_test, labels_test, th, th0)/data_test.shape[1]

```

## 9.2) Evaluating a learning algorithm using a data source

Construct a testing procedure that takes a learning algorithm and a data source as input and runs the learning algorithm multiple times, each time evaluating the resulting classifier as above. It should report the overall average classification accuracy.

You can use our implementation of `eval_classifier` as above.

Write the function `eval_learning_alg` that takes:

- `learner` - a function, such as `perceptron` or `averaged_perceptron`
- `data_gen` - a data generator, call it with a desired data set size; returns a tuple (data, labels)
- `n_train` - the size of the learning sets
- `n_test` - the size of the test sets
- `it` - the number of iterations to average over

and returns the average classification accuracy as a float between 0. and 1..

**Note:** Be sure to generate your training data and then testing data in that order, to ensure that the pseudorandomly generated data matches that in the test code.



```

1 import numpy as np
2
3 def eval_learning_alg(learner, data_gen, n_train, n_test, it):
4     classifier_performance = []
5
6     for i in range(it):
7         train_data, train_labels = data_gen(n_train)
8         test_data, test_labels = data_gen(n_train)
9
10
11         classifier_performance.append(eval_classifier(learner, train_data, train_labels, test_data, test_labels))
12
13     return(sum(classifier_performance)/it)

```

[Ask for Help](#)

100.00%

You have infinitely many submissions remaining.

Here is the solution we wrote:

```

import numpy as np
def eval_learning_alg(learner, data_gen, n_train, n_test, it):
    score_sum = 0
    for i in range(it):
        data_train, labels_train = data_gen(n_train)
        data_test, labels_test = data_gen(n_test)
        score_sum += eval_classifier(learner, data_train, labels_train,
                                    data_test, labels_test)
    return score_sum/it

```

### 9.3) Evaluating a learning algorithm with a fixed dataset

Cross-validation is a strategy for evaluating a learning algorithm, using a single training set of size  $n$ . Cross-validation takes in a learning algorithm  $L$ , a fixed data set  $D$ , and a parameter  $k$ . It will run the learning algorithm  $k$  different times, then evaluate the accuracy of the resulting classifier, and ultimately return the average of the accuracies over each of the  $k$  "runs" of  $L$ . It is structured like this:

```

divide D into k parts, as equally as possible; call them D_i for i == 0 .. k-1
# be sure the data is shuffled in case someone put all the positive examples first in the data!
for j from 0 to k-1:
    D_minus_j = union of all the datasets D_i, except for D_j
    h_j = L(D_minus_j)
    score_j = accuracy of h_j measured on D_j
return average(score_0, ..., score_(k-1))

```

So, each time, it trains on  $k-1$  of the pieces of the data set and tests the resulting hypothesis on the piece that was not used for training.

When  $k = n$ , it is called *leave-one-out cross validation*.

Implement cross validation **assuming that the input data is shuffled already** so that the positives and negatives are distributed randomly. If the size of the data does not evenly divide by  $k$ , split the data into  $n \% k$  sub-arrays of size  $n//k + 1$  and the rest of size  $n//k$ . (Hint: You can use [numpy.array\\_split](#) and [numpy.concatenate](#) with axis arguments to split and rejoin the data as you desire.)

Note: In Python,  $n//k$  indicates integer division, e.g.  $2//3$  gives 0 and  $4//3$  gives 1.

```

1 import numpy as np
2
3 def xval_learning_alg(learner, data, labels, k):
4
5     if data.shape[1]%k != 0:
6         k+=1
7
8     split = [to_fix.T for to_fix in np.array_split(data.T,k)]
9     labels_split = [to_fix.T for to_fix in np.array_split(labels.T, k)]
10
11     scores = []
12
13     for iter in range(k):

```

Ask for Help

100.00%

You have infinitely many submissions remaining.

Here is the solution we wrote:

```

import numpy as np
def xval_learning_alg(learner, data, labels, k):
    s_data = np.array_split(data, k, axis=1)
    s_labels = np.array_split(labels, k, axis=1)

    score_sum = 0
    for i in range(k):
        data_train = np.concatenate(s_data[:i] + s_data[i+1:], axis=1)
        labels_train = np.concatenate(s_labels[:i] + s_labels[i+1:], axis=1)
        data_test = np.array(s_data[i])
        labels_test = np.array(s_labels[i])
        score_sum += eval_classifier(learner, data_train, labels_train,
                                   data_test, labels_test)

    return score_sum/k

```

## 10) Testing

In this section, we compare the effectiveness of perceptron and averaged perceptron on some data that are not necessarily linearly separable.

Use your `eval_learning_alg` and the `gen_flipped_lin_separable` function in the code file to evaluate the accuracy of perceptron vs. averaged perceptron. `gen_flipped_lin_separable` is a wrapper function that returns a generator - `flip_generator`, which can be called with an integer to return a data set and labels. Note that this generates linearly separable data and then "flips" the labels with some specified probability (the argument `pflip`); so most of the results

will not be linearly separable. You can also **specify** `pflip` in the call to the generator wrapper function. At the bottom of the code distribution is an example.

Run enough trials so that you can confidently predict the accuracy of these algorithms on new data from that same generator; assume training/test sets on the order of 20 points. The Tutor will check that your answer is within 0.025 of the answer we got using the same generator.

Accuracy for perceptron (with flip probability 0.1):

[Submit](#)[View Answer](#)[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Accuracy for averaged perceptron (with flip probability 0.1):

[Submit](#)[View Answer](#)[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Accuracy for perceptron (with flip probability 0.25):

[Ask for Help](#)**0.00%**

*You have infinitely many submissions remaining.*

Solution: 0.59

Accuracy for averaged perceptron (with flip probability 0.25):

[Submit](#)[View Answer](#)[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Modify your `eval_learning_alg` so that it tests hypothesis on the training data instead of generating a new test data set. Run enough trials that you can confidently predict this "training accuracy" for the two learning algorithms. Note the differences from your results above.

Accuracy for perceptron (with flip probability 0.1) on training data:

[Submit](#)[View Answer](#)[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Accuracy for averaged perceptron (with flip probability 0.1) on training data:

[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Solution: 0.87

Accuracy for perceptron (with flip probability 0.25) on training data:

[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Solution: 0.66

Accuracy for averaged perceptron (with flip probability 0.25) on training data:

[Ask for Help](#)**100.00%**

*You have infinitely many submissions remaining.*

Solution: 0.72