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** $y^{(i)}$

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.

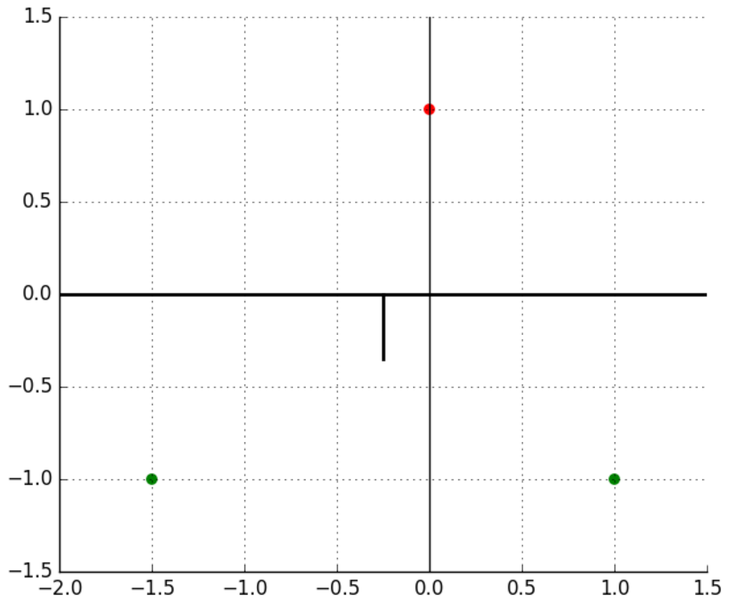
1.1) Take 1

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

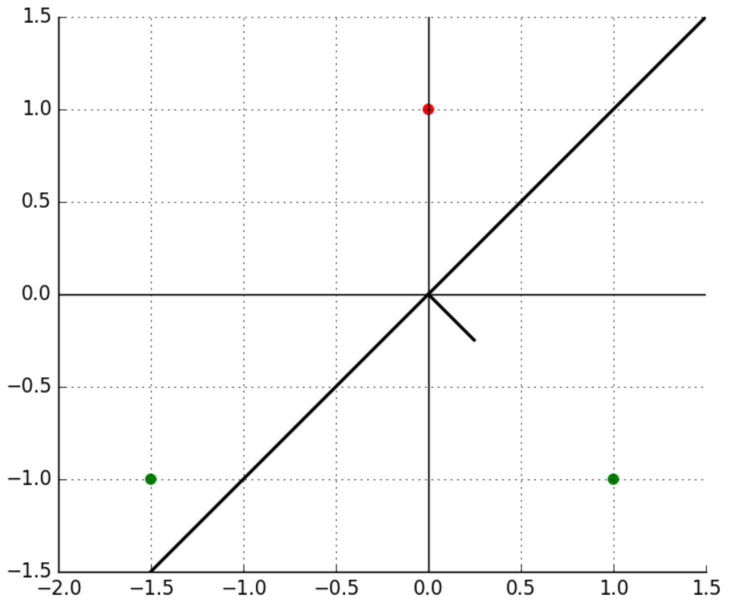
Number of mistakes is: 2

Submit View Answer Ask for Help **100.00%**

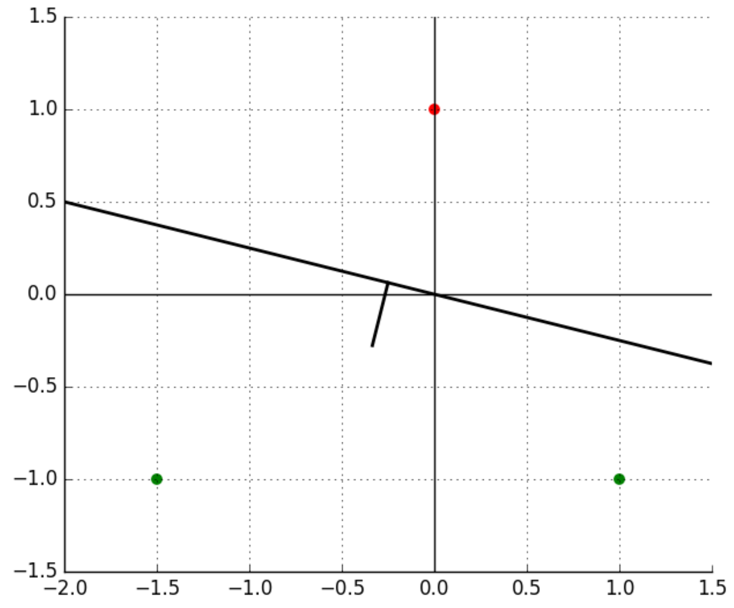
You have infinitely many submissions remaining.

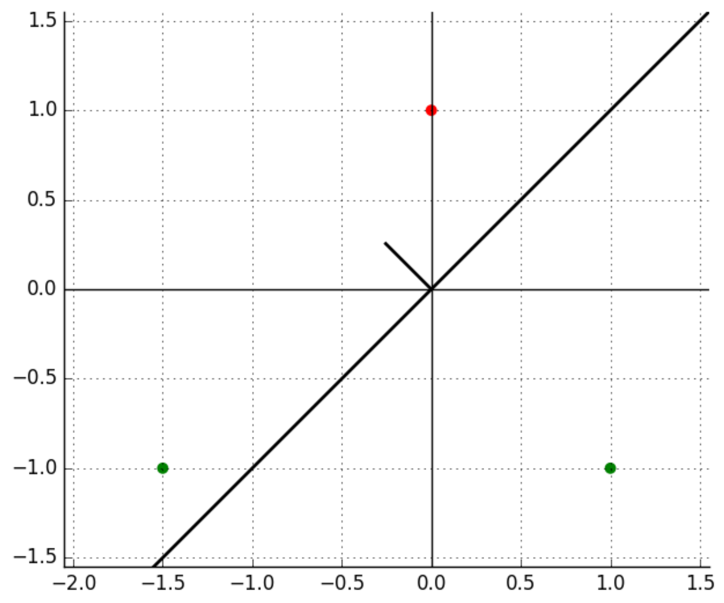

1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

(1)


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

(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 θ changes.

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

[2,3]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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: 1

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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.

[1]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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 6

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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

Number of mistakes is 1

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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 θ is initialized with larger values, corrections to θ have a smaller impact. Therefore, in general, it takes more corrections until θ 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 θ as a Python list or the string `'none'` if none exists.'

[0,-1]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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). θ is initialized to the zero vector and θ_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
4	$[2, 2]$	-1	1
5	$[1, -1]$	-1	0

3.1) What is the post training θ?

Provide it as a python list of the form $[a, b]$. [-2,0]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

3.2) What is the post training θ_0?

Provide it as a number. -5

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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 θ such that, for any $x = [x_1, x_2, x_3]$, where $x_i \in \{0, 1\}$:

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

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

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. [0,0,0,0,0,0,0,1]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

4.1b) Assuming $\theta_0 = 0$ (no offset), enter θ 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'` 'none'

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

4.1c) Assuming θ_0 is non-zero (offset), enter a θ and θ_0 as a Python list of length 4 (θ_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'`. [1,1,1,-2.5]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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.' 'none'

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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 (`[x0_1,x0_2,r]`) or the string `'none'` if none exists.'

[2,2,2]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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

Enter a list with 2 entries for coordinates of θ or the string `'none'` if none exists.'

'none'

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

4.2d) On one side of a line with normal θ and offset θ_0 (recall normal vector points into positive half-space).

Enter a list with 3 entries for coordinates of θ and θ_0 (`[theta_1, theta_2, theta0]`)or the string `'none'` if none exists.'

[-1,-1, 0.5]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

4.2e)

Which of the above are families of linear classifiers?

- [ ] 4.2a
- [ ] 4.2b
- [✓] 4.2c
- [✓] 4.2d

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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

i	**Data Points** $x^{(i)}$	**Labels** $y^{(i)}$
3	$[2, -1]$	1
4	$[2, 1]$	-1

This dataset is: Not linearly separable

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

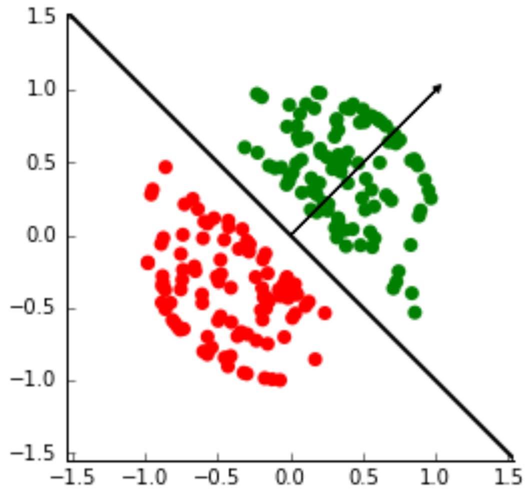
6) Mistakes and generalization

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 γ (the margin of the separator for the dataset). Consider values of R in the range $[1, 10]$ and values of γ in the range $[0.01, 2]$.

6.1a)

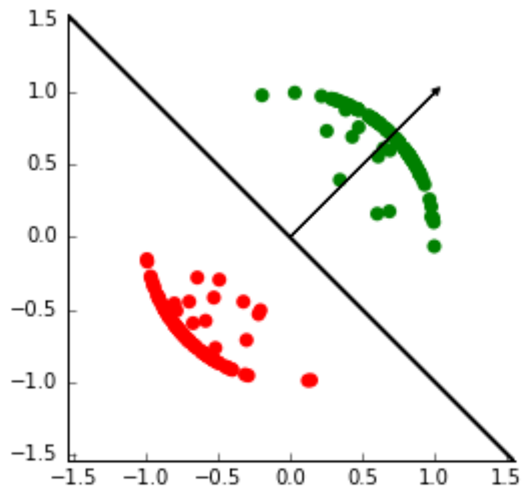
(1) 

Enter a Python list with 2 floats, a value of R and a value of gamma: [2, 0.2]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

6.1b)

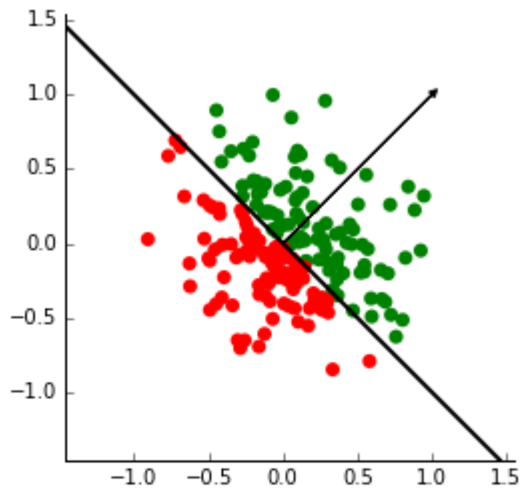
(2) 

Enter a Python list with 2 floats, a value of R and a value of gamma: [1.1, 0.5]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

6.1c)



(3)

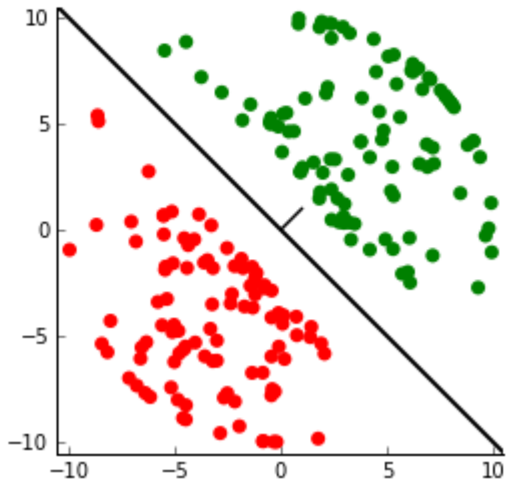
Enter a Python list with 2 floats, a value of R and a value of gamma: [1.3, 0.01]

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

6.1d)

(4)



Enter a Python list with 2 floats, a value of R and a value of gamma: [10, 2]

100.00%

You have infinitely many submissions remaining.

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 γ?

```
(.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. [1e10, 1e8, 1e6, 1e4, 100, 25, 4]

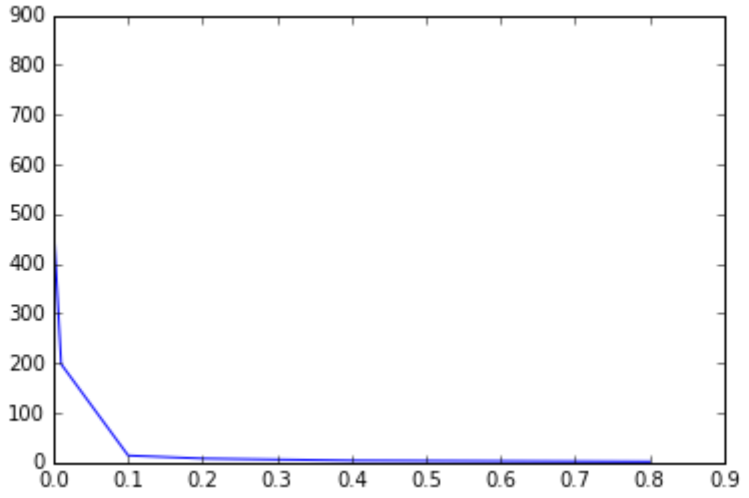
Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

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 γ.

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 `(th, th0)` 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 θ (a d by 1 array) and θ_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 `(th, th0)` (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.**

```python
        for i in range(n):
            xi = data[:,i:i+1]
            yi = labels[0, i]
            if yi * np.sign(theta.T @ xi + theta_0) <= 0:
                founderror = True
                theta += yi * xi
                theta_0 += yi
                if hook:
                    hook(theta, theta_0)
        if not founderror:
            break
    #plot_separator(ax=ax, th=theta, th_0=theta_0)
    return (theta, np.array([[theta_0]]))

```

Run Code Submit View Answer Ask for Help **100.00%**

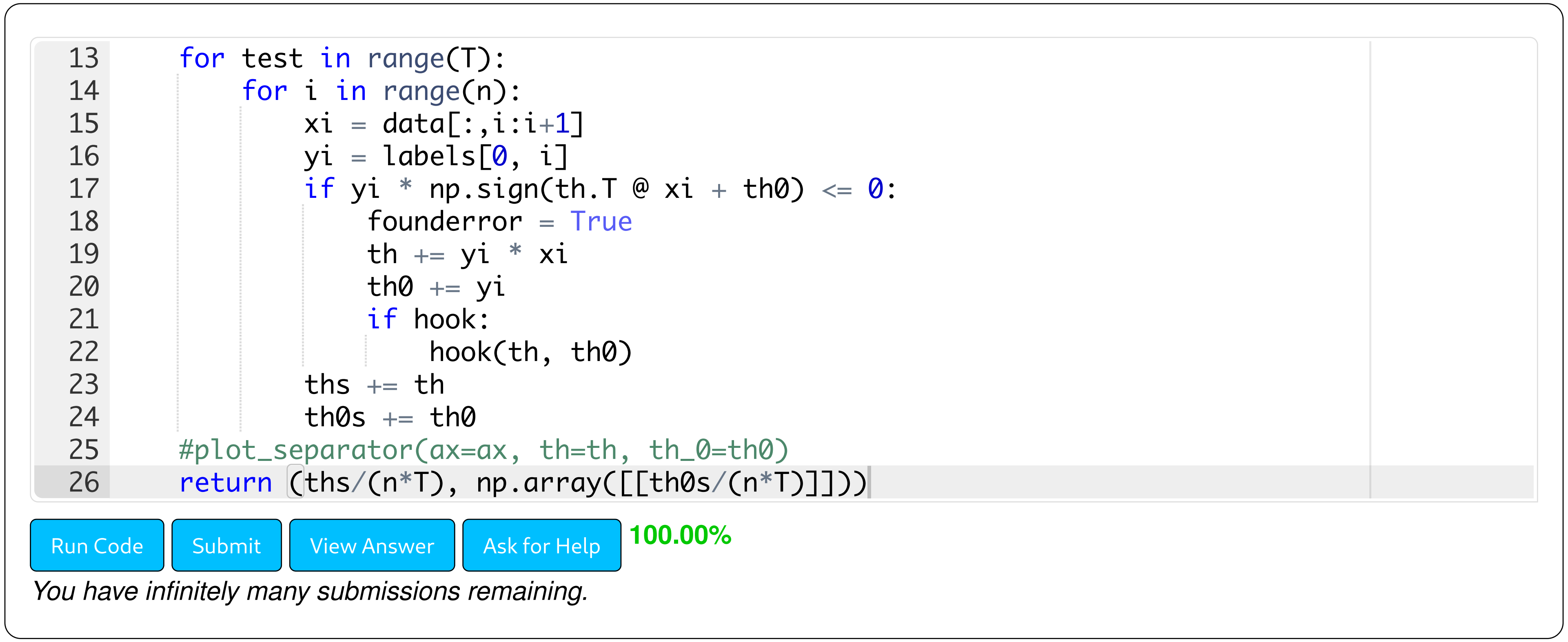
You have infinitely many submissions remaining.

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 `th` and `th0` 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)
```



```python
    for test in range(T):
        for i in range(n):
            xi = data[:,i:i+1]
            yi = labels[0, i]
            if yi * np.sign(th.T @ xi + th0) <= 0:
                founderror = True
                th += yi * xi
                th0 += yi
                if hook:
                    hook(th, th0)
            ths += th
            th0s += th0
    #plot_separator(ax=ax, th=th, th_0=th0)
    return (ths/(n*T), np.array([[th0s/(n*T)]]))
```

You have infinitely many submissions remaining.

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 θ, θ_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..

```python
import numpy as np

def eval_classifier(learner, data_train, labels_train, data_test, labels_test):
    th, th0 = learner(data_train, labels_train)
    n = data_test.shape[1]
    correctnum = score(data_test, labels_test, th, th0)
    return correctnum / n

```

Run Code Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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.

```
    sum_ = 0
    for i in range(it):
        n = n_train + n_test
        x, y = data_gen(n)
        inxs = np.arange(n)
        #np.random.shuffle(inxs)
        data_train = x[:,inxs[:n_train]]
        labels_train = y[:,inxs[:n_train]]
        data_test = x[:,inxs[n_train:]]
        labels_test = y[:,inxs[n_train:]]
        sum_ += eval_classifier(learner, data_train, labels_train, data_test, labels_test)
    return sum_/ it

```

Run Code Submit View Answer Ask for Help 100.00%

You have infinitely many submissions remaining.

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 $\mathcal{D}$, 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(score0, ..., 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.

```python
def xval_learning_alg(learner, indata, inlabels, k):
    ds = np.split(indata, k, axis=1)
    ls = np.split(inlabels, k, axis=1)
    scores = []
    for j in range(k):
        data = np.concatenate(ds[:j] + ds[j+1:], axis=1)
        labels = np.concatenate(ls[:j] + ls[j+1:], axis=1)
        th, th0 = learner(data, labels)
        correctnum = score(ds[j], ls[j], th, th0)
        n = len(ls[j][0])
        print(correctnum, n)
        scores += [correctnum * 1. / n]
    return np.average(scores)

```

Run Code Submit View Answer Ask for Help 100.00%

You have infinitely many submissions remaining.

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 **specifiy** `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): 0.7590500000000006

Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

Accuracy for averaged perceptron (with flip probability 0.1): 0.8048499999999997

Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

Accuracy for perceptron (with flip probability 0.25): 0.5873500000000001

Submit | View Answer | Ask for Help **100.00%**

You have infinitely many submissions remaining.

Accuracy for averaged perceptron (with flip probability 0.25): 0.6375000000000007

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: 0.8195

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:

0.8665999999999997

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

Accuracy for perceptron (with flip probability 0.25) on training data: 0.6714499999999998

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.

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

0.7329499999999997

Submit View Answer Ask for Help **100.00%**

You have infinitely many submissions remaining.