MIT 6.036 Spring 2019: Homework 2

This colab notebook provides code and a framework for problems 7-10 of the homework. You can work out your solutions here, then submit your results back on the homework page when ready.

Setup

First, download the code distribution for this homework that contains test cases and helper functions (such as `positive`).

Run the next code block to download and import the code for this lab.

In [3]:
```
!rm -f code_for_hw02.py*
!wget --no-check-certificate --quiet https://introml_oll.odl.mit.edu/6.036/static/homework/hw02/code_for_hw02.py
from code_for_hw02 import *
```

```
Importing code_for_hw02
New procedures added: tidy_plot, plot_separator, plot_data, plot_nonlin_sep, cv,
                      rv, y, positive, score
Data Sets: super_simple_separable_through_origin(), super_simple_separable(), xor(),
           xor_more()
Test data for problem 2.1: data1, labels1, data2, labels2
Test data for problem 2.2: big_data, big_data_labels, gen_big_data(), gen_lin_separable(),
                           big_higher_dim_separable(), gen_flipped_lin_separable()
Test functions: test_linear_classifier(), test_perceptron(), test_averaged_perceptron(),
                test_eval_classifier(), test_eval_learning_alg(), test_xval_learning_alg()

For more information, use 'help', e.g. 'help tidy_plot'
Done with import of code_for_hw02
```

In [4]:
```
help(tidy_plot)
```

```
Help on function tidy_plot in module code_for_hw02:

tidy_plot(xmin, xmax, ymin, ymax, center=False, title=None, xlabel=None, ylabel=None)
    Set up axes for plotting
    xmin, xmax, ymin, ymax = (float) plot extents
    Return matplotlib axes
```

In [5]:
```
def test(a):
  return a + 53
```

In [6]:
```
def methodB(a):
  return test(a)
```

In [7]:
```
def someMethod():
  test = 7
  return methodB(test + 3)
```

In [8]:
```
someMethod()
```

Out[8]:
```
63
```

7) Implement perceptron

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.

In [9]:
```
import numpy as np
```

In [248…

```
def perceptron(data, labels, params={}, hook=None):
    # if T not in params, default to 100
    T = params.get('T', 100)
    d = data.shape[0]
    n = data.shape[1]
    theta = np.transpose([[0.0] * d])
    theta_0 = 0.0
    #ax = plot_data(data, labels)
    for test in range(T):
        founderror = False
        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]]))
```

In [249…

```
test_perceptron(perceptron)
```

```
-----------Test Perceptron 0-----------
Passed!

-----------Test Perceptron 1-----------
Passed!
```

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)
```

In [367…

```python
import numpy as np

def averaged_perceptron(data, labels, params={}, hook=None):
    # if T not in params, default to 100
    T = params.get('T', 100)
    d = data.shape[0]
    n = data.shape[1]
    th = np.transpose([[0.0] * d])
    th0 = 0.0
    ths = np.transpose([[0.0] * d])
    th0s = 0.0
    #ax = plot_data(data, labels)
    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)]]))
```

In [251…

```python
test_averaged_perceptron(averaged_perceptron)
```

```
-----------Test Averaged Perceptron 0-----------
Passed!

-----------Test Averaged Perceptron 1-----------
Passed!
```

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.

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

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 1 by 1 matrix with an integer indicating number of data points correct for the separator.

In [252…

```
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
```

In [253…

```
test_eval_classifier(eval_classifier,perceptron)
```

```
-----------Test Eval Classifier 0-----------
Passed!

-----------Test Eval Classifier 1-----------
Passed!
```

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.

In [254…

```
import numpy as np

def eval_learning_alg(learner, data_gen, n_train, n_test, it):
    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
```

In [255…

```
test_eval_learning_alg(eval_learning_alg,perceptron)
```

```
-----------Test Eval Learning Algo-----------
Passed!
```

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.

In [291…

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6]])
```

In [292…

```
np.concatenate((a, b), axis=0), np.concatenate((a, b.T), axis=1),  np.concatenate((a, b), axis=None)
```

Out[292]:
```
(array([[1, 2],
        [3, 4],
        [5, 6]]),
 array([[1, 2, 5],
        [3, 4, 6]]),
 array([1, 2, 3, 4, 5, 6]))
```

In [347…
```python
import numpy as np

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)
```

In [348…
```python
test_xval_learning_alg(xval_learning_alg,perceptron)
```

```
11 20
12 20
10 20
13 20
15 20
-----------Test Cross-eval Learning Algo-----------
Passed!
```

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` generator in the code file to evaluate the accuracy of `perceptron` vs. a`veraged_perceptron`. `gen_flipped_lin_separable` 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. You should use the default values of th and th_0 to retain consistency with the Tutor.

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.

In [358…
```
print(eval_learning_alg(perceptron, gen_flipped_lin_separable(pflip=.1), 20, 20, 1000))
```
```
0.7590500000000006
```

In [359…
```
print(eval_learning_alg(averaged_perceptron, gen_flipped_lin_separable(pflip=.1), 20, 20, 1000))
```
```
0.8048499999999997
```

In [360…
```
print(eval_learning_alg(perceptron, gen_flipped_lin_separable(pflip=.25), 20, 20, 1000))
```
```
0.5873500000000001
```

In [361…
```
print(eval_learning_alg(averaged_perceptron, gen_flipped_lin_separable(pflip=.25), 20, 20, 1000))
```
```
0.6375000000000007
```

In [362…

```
def eval_learning_alg_wrong(learner, data_gen, n_train, n_test, it):
    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_train, labels_train)
    return sum_/ it
```

In [363…

```
print(eval_learning_alg_wrong(perceptron, gen_flipped_lin_separable(pflip=.1), 20, 20, 1000))
```

```
0.8195
```

In [364…

```
print(eval_learning_alg_wrong(averaged_perceptron, gen_flipped_lin_separable(pflip=.1), 20, 20, 1000))
```

```
0.8665999999999997
```

In [365…

```
print(eval_learning_alg_wrong(perceptron, gen_flipped_lin_separable(pflip=.25), 20, 20, 1000))
```

```
0.6714499999999998
```

In [366…

```
print(eval_learning_alg_wrong(averaged_perceptron, gen_flipped_lin_separable(pflip=.25), 20, 20, 1000))
```

```
0.7329499999999997
```

In [ ]: