Homework 2

The questions below are due on Sunday September 24, 2017; 11:00:00 PM.

You are not logged in.

If you are a current student, please Log In (https://introml.mit.edu/fall17/homework/hw02?loginaction=login) for full access to this page.

A code file that will be useful in later problems can be found here (https://introml.mit.edu/__STATIC__/fall17/homework/hw02/code_for_hw2.py.zip).

1) XOR

Consider this data-set of four points in two-dimensional space:

```
data = ([[1, 1, 2, 2],
         [1, 2, 1, 2]])
labels = [[-1, 1, 1, -1]]
```

It is standardly called the "exclusive-or" problem.

1.1) Part 1

Find a function $\phi : R^2 \rightarrow R^D$ for some dimension D such that, if we were to transform each example $x^{(i)}$ into $\phi(x^{(i)})$, the resulting data set would be linearly separable.

Provide your function as a Python procedure that takes in a (column vector) data point $x^{(i)}$ from `data` and outputs the transformed data point $\phi(x^{(i)})$ (also as a column vector).

```
import numpy as np

def phi(x_i):
    return np.vstack((x_i, x_i**2))

```

Ask for Help

1.2) Part 2

Provide the parameter vector θ and offset θ_0 that constitute a linear separator of the data when transformed according to your f.

Enter a Python list of the form `[transformed_data, th, th0]` where `transformed_data` is the list form of the (a d by 4) data as transformed by your function, `th` is a list of d floats corresponding to θ and `th0` is a float corresponding to θ_0.

in hw2.py

Ask for Help

2) SCALING

Consider a linearly separable dataset with two features:

```
data = ([[200, 800, 200, 800],
         [.2,  .2,  .8,  .8]])
labels = [[-1, -1, 1, 1]]
```

Consider the separator defined by $\theta = (0, 1), \theta_0 = -0.5$.

To apply the perceptron mistake bound, think of this as a perceptron through the origin with $\theta = (0, 1, -0.5)$ and θ_0 = 0.

For a separator through the origin, recall that the margin is the minimum of $\gamma = y^{(i)}(\theta^T x^{(i)})/\|\theta\|$ over all data points $(x^{(i)}, y^{(i)})$.

1.

What is the margin of this data set with respect to that separator?

0.26832816

Ask for Help

2.

What is the theoretical bound on the number of mistakes perceptron will make on this problem? 8888911.49

Ask for Help

3.

Roughly how many mistakes does perceptron through origin have to make in order to find a perfect separator? (Try it)

833376

Ask for Help

4.

If we were to multiply both original features features of all of the points by $.001$, and considered the separator through origin $\theta = (0, 1, -0.0005)$, what would the margin of the new dataset be? 0.0003

Ask for Help

5.

How would the performance of the perceptron (as predicted by the mistake bound) change?

More mistakes

Ask for Help

6.

If we multiplied just the first original feature by .001, and used our original separator, what would the new margin be?

0.26832816

Ask for Help

7.

What would the mistake bound be in this case?

31.66666667

Ask for Help

8.

Run the perceptron algorithm on this data; how many mistakes does it make?

16

Ask for Help

3) ENCODING DISCRETE VALUES

Some data sets have features that take on discrete values drawn from a set. Examples might be:

- which section of a class a student is in (1, 2, 3, 4)
- manufacturer of a cell phone (Samsung, Xiaomi, Sony, Apple, LG, Nokia)
- which laboratory performed a particular medical test

Sometimes they already have an obvious encoding into integers; other times, they don't but it's easy to make one (e.g., Samsung = 1, Xiaomi = 2, Sony = 3, Apple = 4, LG = 5, Nokia = 6)

1) Let's consider the case of the cell phones, using the encoding above, and imagine there is some prediction problem for which we have the data set:

```
data =   [[2, 3,  4,  5]]
labels = [[1, 1, -1, -1]]
```

What value of θ and θ_0 would we get when running perceptron on this data?

Enter a Python list with two floats, one for $heta$ and one for $heta_0$.

[-2., 7.]

Ask for Help

2) What prediction would we make about other phone types based on this classifier?

Enter a Python list with two labels (1 or -1), the first one for a Samsung phone and the second for a Nokia phone. [1, -1]

Ask for Help

3)

Are those predictions well justified by the data? Yes

Ask for Help

4) It is common to encode sets of discrete values, for machine learning, not as a single multi-valued feature, but using a *one hot* encoding. So, if there are k values in the discrete set, we would transform that single multi-valued feature into k binary-valued features, in which feature i has value $+1$ if the original feature value was i and has value 0 (or -1) otherwise.

Write a function `one_hot` that takes as input x, a single feature value, and k, the total possible number of values (from 1 to k) this feature can take on, and transform it to a column vector of k binary features using a one-hot encoding.

```
import numpy as np

#[1,2,3...k]
def one_hot(x, k):
    col_vector = np.zeros((k,1)); col_vector[x-1] = 1
    return col_vector

```

Ask for Help

5) What happens if we use one-hot encoding on the data set from the first part and put it into the perceptron? Recall that for a classifier $h(x)$, the prediction is $+1$ if $h(x) > 0$ and -1 otherwise.

a) What is the separator?

Enter a Python list with 7 floats, six for θ and one for θ_0.

`[0., 2., 1.,-2.,-1., 0., 0.]`

Ask for Help

b) What are the predictions for Samsung and Nokia?

Enter a Python list with two labels (1 or -1), the first one for a Samsung phone and the second for a Nokia phone. `[0,0]`

Ask for Help

c) What are the distances for the Samsung and Nokia data points from the separator?

Enter a Python list with two distances, the first one for a Samsung phone and the second for a Nokia phone. `[0,0]`

Ask for Help

6) Now, what if we have this dataset:

```
data =   [[1, 2, 3, 4, 5, 6]]
labels = [[1, 1, -1, -1, 1, 1]]
```

Is it linearly separable in the original encoding? No

Ask for Help

7) Is it linearly separable in the one-hot encoding? If so, provide the separator found by the perceptron.

Enter a Python list with 7 floats, six for θ and one for θ_0 or `'none'`

```
[ 1. , 1. ,-2. ,-2. , 1., 1. ,0]
```

Ask for Help

8) Enter an assignment of data values to labels that is not linearly separable using the one-hot encoding, or enter None if no such assignment exists.

Enter a Python list with 6 tuples `(value, label)` or `'none'`

```
'none'
```

Ask for Help

3.1) Feature Vectors

Consider a sequence of n-dimensional data points, $x^{(1)}, x^{(2)}$, ..., and a sequence of m-dimensional feature vectors, $z^{(1)}, z^{(2)}$, ..., extracted from the x's by a linear transformation, $z^{(i)} = Ax^{(i)}$. If m is much smaller than n, you might expect that it would be easier to learn in the lower dimensional feature space than in the original data space.

(a) Suppose $n = 6$, $m = 2$, z_1 is the average of the elements of x, and z_2 is the average of the first three elements of x minus the average of fourth through sixth elements of x.

What is A? Enter your solution as a list of lists. (e.g: [[row1],[row2]])

```
[[1/6]*6, [1/3]*3+[-1/3]*3]
```

Ask for Help

(b) Suppose $h(z) = sign(\theta_z \cdot z)$ is a classifier for the feature vectors, and $h(x) = sign(\theta_x \cdot x)$ is a classifier for the original data vectors.

Given a θ_z that produces good classifications of the feature vectors, is there a θ_x that will identically classify the associated x's? In fact, such that $\theta_z \cdot z = \theta_x \cdot x$. Yes

Ask for Help

If so, provide an expression for θ_x in terms of A and θ_z.

Enter your answer as a Python expression. You can use `theta_z` to stand for θ_z, `x` to stand for any array x, `transpose(x)` for transpose of an array, `norm(x)` for the length(norm) of a vector, and `x@y` to indicate a matrix product of two arrays.

```
transpose(A)@theta_z
```

Ask for Help

(c)

Given the same classifiers as in (b), if there is a θ_x that produces good classifications of the data vectors, will there always be a θ_z that will identically classify the associated z's? No

Ask for Help

(d)

If $m < n$, can the perceptron (through the origin) algorithm converge more quickly (make fewer mistakes) when training in z-space? Yes

Ask for Help

If so, provide an example. Consider the case with $m = 1$ and $n = 2$ and only two data points. You'll need to specify matrix $A = [a, b]$, and one point $x^{(1)} = [c, d]^T$ that we will assume has label -1, and a second point $x^{(2)} = [e, f]^T$ that we will assume has the label +1.

Enter a Python list of 6 numbers, `[a, b, c, d, e, f]` or `'none'`

```
[should exist but ican't find it..
```

Ask for Help

(e)

If $m < n$, can we find a more accurate classifier by training in z-space, as measured on the training data? No

Ask for Help

How about on unseen data? Yes

Ask for Help

4) POLYNOMIAL FEATURES

One systematic way of generating non-linear transformations of your input features is to consider the polynomials of increasing order. Given a feature vector $x = [x_1, x_2, ..., x_d]^T$, we can map it into a new feature vector that contains all the factors in a polynomial of order d, for example, for $x = [x_1, x_2]^T$ and order 2, we get

$$\phi(x) = [1, x_1, x_2, x_1x_2, x_1^2, x_2^2]^T$$

and for order 3, we get

$$\phi(x) = [1, x_1, x_2, x_1x_2, x_1^2, x_2^2, x_1^2x_2, x_1x_2^2, x_1^3, x_2^3]^T$$

. In the code file, we have defined `make_polynomial_feature_fun` that, given the order, returns a feature transformation function (analogous to ϕ in the first problem). You should use it in doing this problem.

Enter a list of 6 integers indicating the number of polynomial features of degrees [1, 10, 20, 30, 40, 50] for a 2-dimensional feature vector.

```
[3, 66, 231, 496, 861, 1326]
```

Ask for Help

Enter a list of 6 integers indicating the number of polynomial features of degrees [1, 10, 20, 30, 40, 50] for a 3-dimensional feature vector.

```
[4, 286, 1771, 5456, 12341, 23426]
```

Ask for Help

In the code file, we have defined 4 sample data sets, (1) `super_simple_separable_through_origin`, (2) `super_simple_separable`, (3) `xor`, and (4) `xor_more`. On your own machine, you should run the code we have provided (`test_with_features`) for various orders of polynomial features and enter below the order of the smallest feature that separates the data. Make sure that you have included your implementation of `perceptron` in that file. You may need to adjust the number of iterations that the perceptron runs.

The separators are displayed when the code runs; it's instructive to watch them to see the range of separators that these non-linear transformations produce. Note that the separators are drawn by evaluating the feature transformations on a grid of points in the feature space and using the separator to classify them.

Enter a Python list of integers indicating the smallest polynomial order for which a separator exists for each of the four datasets in the code file (in order).

```
[1, 1, 2, 3]
```

Ask for Help