Week 0 Lab

The questions below are due on Sunday September 10, 2017; 11:00:00 PM.

You are not logged in.

If you are a current student, please Log In (https://introml.mit.edu/fall17/labs/lab00?loginaction=login) for full access to this page.

This lab is divided into three sections:

- An introduction to numpy
- Some linear algebra relevant to machine learning
- Some numpy practice relevant to machine learning

1) NUMPY INTRO

`numpy` is a package for doing a variety of numerical computations in Python. We will use it extensively. It supports writing very compact and efficient code for handling arrays of data. We will start every code file that uses `numpy` with `import numpy as np`, so that we can reference numpy functions with the 'np.' precedent.

You can find general documentation on `numpy` here (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html).

The fundamental data type in numpy is the multidimensional array, and arrays are usually generated from a nested list of values using the `np.array` command. Every `array` has a `shape` attribute which is a tuple of dimension sizes.

In this class, we will use two-dimensional arrays almost exclusively. That is, we will use 2D arrays to represent both matrices and vectors! This is one of several times where we will seem to be unnecessarily fussy about how we construct and manipulate vectors and matrices, but we have our reasons. We have found that using this format results in predictable results from numpy operations.

Using 2D arrays for matrices is clear enough, but what about column and row vectors? We will represent a column vector as a $d \times 1$ array and a row vector as a $1 \times d$ array. So for example, we will represent the three-element column vector,

$$x = \begin{bmatrix} 1 \\ 5 \\ 3 \end{bmatrix},$$

as a 3×1 numpy array. This array can be generated with

```
x = np.array([[1],[5],[3]]),
```

or by using the transpose of a 1×3 array (a row vector) as in,

```
x = np.transpose(np.array([[1,5,3]]),
```

where you should take note of the "double" brackets.

It is often more convenient to use the array attribute `.T` , as in

```
x = np.array([[1,5,3]]).T.
```

to compute the transpose.

Before you begin lab, we would like to note that we do not like "loops", and will not accept answers that use them. One reason for avoiding loops is efficiency. For many operations, numpy calls a compiled library written in C, and the library is far faster that interpreted Python (in part due to the low-level nature of C, optimizations like vectorization, and in some cases, parallelization). But the more important reason for avoiding loops is that using higher-level constructs leads to simpler code that is easier to debug. So, we expect that you should be able to transform loop operations into equivalent operations on numpy arrays, and we will practice this in today's lab.

1.1) Array

Provide an expression that sets `A` to be a 2×3 `numpy` array, containing any values you wish.

```
1 import numpy as np
2 A = np.array([[1,2,3], [4,5,6]])
3
```

Ask for Help

1.2) Transpose

Write a procedure that takes an array and returns the transpose of the array. You can use 'np.transpose' or the '.T', but you may not use a loop.

```
1 import numpy as np
2 def tr(A):
3     return A.T
4
```

Ask for Help

1.3) Shapes

Let `A` be a 4×2 array, `B` be 4×3, `C` be 4×1. For each of the following expressions, indicate the shape of the result as a tuple of integers or "none" (as a Python string with quotes) if it is illegal.

1.

```
C * C
```

```
(4,1)
```

Ask for Help

2.

```
np.dot(C, C)
```

```
'none'
```

Ask for Help

3.

```
np.dot(np.transpose(C), C)
```

```
(1,1)
```

Ask for Help

4.

```
np.dot(A, B)
```

```
'none'
```

Ask for Help

5.

```
np.dot(np.transpose(A), B)
```

```
(2,3)
```

Ask for Help

1.4) Row vector

Write a procedure that takes a list of numbers and returns a 2D numpy array representing a row vector containing those numbers.

```
1 import numpy as np
2 def rv(value_list):
3     return np.array([value_list])
4
```

Ask for Help

1.5) Column vector

Write a procedure that takes a list of numbers and returns a 2D numpy array representing a column vector containing those numbers. You can use the `rv` procedure.

```
1 import numpy as np
2 def cv(value_list):
3     return rv(value_list).T
4
```

Ask for Help

1.6) length

Write a procedure that takes a column vector and returns the vector's Euclidean length (or equivalently, its magnitude) as a scalar. You may not use `np.linalg.norm`, and you may not use a loop.

```
1 import numpy as np
2 def length(col_v):
3     return np.sqrt(np.sum(col*col))
4
```

Ask for Help

1.7) normalize

Write a procedure that takes a column vector and retuns a unit vector in the same direction. You may not use a for loop. Use your `length` procedure from above (you do not need to define it again).

```
1 import numpy as np
2 def normalize(col_v):
3     return (1/length(col_v))*col_v
4
```

Ask for Help

2) HYPERPLANES

2.1) Through origin

In an d-dimensional space, a hyperplane is a $d-1$ dimensional subspace that can be characterized by a scalar offset, and a normal to the subspace. For example, any line is a hyperplane in two dimensions, an infinite flat sheet is a hyperplane in three dimensions, but in higher dimensions, hyperplanes are harder to visualize. Fortunately, they are easy to specify.

In d dimensions, any vector $\theta \in R^d$ can define a hyperplane. Specifically, the hyperplane associated with θ is the set of all vectors $x \in R^d$ such that $\theta^T x = 0$. Note that this hyperplane includes the origin, since $x = 0$ is in the set.

1. In two dimensions, $\theta = [\theta_1, \theta_2]$ can define a hyperplane. Let $\theta = [1, 2]$, and determine a vector that is normal to the hyperplane given by the set of all $x \in R^2$ such that $\theta^T x = 0$. Enter your answer as a Python list of numbers.

   Vector normal to hyperplane

   ```
   [1,2]
   ```

   Ask for Help

2. Now, in d dimensions, supply the formula for a unit vector normal to the hyperplane specified by θ where $\theta \in R^d$.

Here are some optional steps that build up to the answer:

- Consider two points P_1, P_2 on the hyperplane and let $U = P_1 - P_2$. What is the value of $\theta^T U$?

- Remembering the formula for a dot product $\theta \cdot U = \theta^T U = \|\theta\| \cdot \|U\| \cos \alpha$, what is the angle between U and θ?

In this question and the subsequent ones that ask for a formula, enter your answer as a Python expression. You can use `theta` to stand for θ, `theta_0` to stand for θ_0, `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.

```
theta * (1/norm(theta)
```

Ask for Help

2.2) General hyperplane, distance to origin

Now, we'll consider hyperplanes defined by $\theta^T x + \theta_0 = 0$, which do not necessarily go through the origin.

Define the *positive* side of a hyperplane to be the half-space defined by $\{x \mid \theta^T x + \theta_0 > 0\}$.

1. In two dimensions, let $\theta = [3, 4]$ and $\theta_0 = 5$. What is the signed perpendicular distance from the hyperplane to the origin? The distance should be positive if the origin is on the positive side of the hyperplane, 0 on the hyperplane and negative otherwise.

Distance = 1

Ask for Help

2. Now, in d dimensions, supply the formula for the signed perpendicular distance from a hyperplane specified by θ, θ_0 to the origin.

Here are some optional steps to build up to the main answer:

- Let P be the point that is the perpendicular projection of the origin onto the hyperplane. What is the distance from P to the origin?
- What is the angle between P and θ?
- What is a formula for $\theta^T P$?
- Let the sign of this distance be positive if the origin is on the positive side of the hyperplane.

```
theta_0/norm(theta)
```

Ask for Help

2.3) General hyperplane, distance to point

Now, let x be an arbitrary point in R^d. Give a formula for the signed perpendicular distance from x to the hyperplane specified by θ, θ_0.

Here are some optional steps to build up to the main answer:

- Let P be the point that is the perpendicular projection of x onto the hyperplane, and r be the distance from x to P (r is our quantity of interest). Give a formula for $x - P$ in terms of r and θ.
- Multiply that formula through by θ and find an expression for r.
- Use the fact that P is actually on the hyperplane to remove it from the previous formula, ending up with a formula for the distance that depends only on x, θ, θ_0.

```
(np.dot(theta.T, x) + theta_0)/norm(theta)
```

Ask for Help

2.4) Code for signed distance

Write a python function using numpy operations (no loops!) that takes column vectors (d by 1) `x` and `th` (of the same dimension) and scalar `th_0` and returns the signed perpendicular distance (as a 1 by 1 array) from `x` to the hyperplane encoded by `th, th_0`.

```
1 import numpy as np
2 def signed_dist(x, th, th_0):
3     return (np.dot(th.T, x) + th_0)/norm(th)
4
```

Ask for Help

2.5) Code for side of hyperplane

Write a python function that takes column vectors `x` and `th` of the same dimension and scalar `th_0` and returns `+1` if `x` is on the positive side of the hyperplane encoded by `th, th_0`, `0` if on the hyperplace and `-1` otherwise. The answer should be an array (look at the `np.sign` function). Note that you are allowed to use any functions defined above.

```
1 import numpy as np
2 def positive(x, th, th_0):
3     return np.sign(signed_dist(x, th, th_0))
```

Ask for Help

3) SEPARATORS

Now, given a hyperplane, and a set of data points, we can think about which points are on which side of the hyperplane. This is something we do in many machine-learning algorithms, as we will explore soon. It is also a chance to begin using numpy on larger chunks of data.

3.1) Expressions

We have defined `data` to be a 2 by 5 array (two rows, five columns) of scalars. It represents 5 data points in two dimensions. We have also defined `labels` to be a 1 by 5 array (1 row, five columns) of 1 and -1 values.

```
data = np.transpose(np.array([[1, 2], [1, 3], [2, 1], [1, -1], [2, -1]]))
labels = rv([-1, -1, +1, +1, +1])
```

Provide an expression that sets `A` to each of the quantities below. Only a single relatively short expression is needed for each one. No loops! You can use (our version) of the procedures defined above. Those functions if written purely as matrix operations should work with a data array, not just a single column vector as the first argument, with no change.

1. A 1 by 5 array of values, either +1, 0 or -1, indicating, for each point in `data`, whether it is on the positive side of the hyperplane `theta = [1, 1], theta_0 = -2` .

```
1 import numpy as np
2 A = np.apply_along_axis(positive, 0, data, theta, theta_0 )
3
```

Ask for Help

2. A 1 by 5 array of values, either `True` or `False`, indicating for each point in `data` and corresponding label in `labels` is correctly classified. That is, whether the side of the hyperplane hyperplane `theta = [1, 1], theta_0 = -2` that the point is on, agrees with the specified label.

```
1 import numpy as np
2 A = lables == np.apply_along_axis(positive, 0, data, theta, th
```

Ask for Help

3.2) Score

Write a procedure that takes as input

- `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
- `th_0`: a single scalar or 1 by 1 array, represents a hyperplane

and returns the number of points for which the label is equal to the output of the `positive` function on the point.

Note that `numpy` treats `False` as 0 and `True` as 1, so you can take the sum of a collection of Boolean values directly.

```
1 import numpy as np
2 def score(data, labels, th, th_0):
3     A = lables == np.apply_along_axis(positive, 0, data, th, th_0)
4     return np.sum(A)
5
```

Ask for Help

3.3) Best separator

We have defined a variable `thetas`, a 2 by 10 array of scalars and a variable `theta_0s` a 1 by 10 array of scalars. There are intended to represent 10 possible hyperplanes in two dimensions.

Write a procedure that takes as input

- `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
- `ths`: a `d` by `m` array of floats that together with
- `th0s`: a `1` by `m` array of floats, represents `m` hyperplanes in `d` dimensions

and finds the hyperplane with the highest score on the data and labels. In case of a tie, return the first hyperplane with the highest score, in the form of a tuple of a `d` by 1 array and an offset in the form of 1 by 1 array.

The function `score` that you wrote above was for a single hyperplane separator, think about how to generalize it to multiple hyperplanes and include this modified (if necessary) definition of `score` in the answer box.

```
 1 import numpy as np
 2
 3 # still no loops????
 4 def score(data, labels, ths, th_0s):
 5     output = []
 6     for col in range(ths.shape[1]):
 7         A = lables == np.apply_along_axis(positive, 0, data, ths|
 8         output.append((np.sum(A), col))
 9     return output
10
11 def best_separator(data, labels, ths, th0s):
12     best_score, col = max(score(data, labels, ths, th0s), key = ]
13     return (ths[:,col], th_0s[:,col])
14
```

Ask for Help