Python

Options

- Run on your own computer
- Run on linux machine in OLVR 329
- Connect to studentweb.cmix.louisiana.edu
  - From your own computer
  - From a linux machine in OLVR 329
  - if (OS == Win10) {
    search for the command prompt app;
    ssh ULID@studentweb.cmix.louisiana.edu
    }
  - if (OS == linux || OS == macos) {
    open terminal;
    ssh ULID@studentweb.cmix.louisiana.edu
    }

If you are not able to log in using your ULID, try your CLID

Make sure Python version is >= 2.7

Python Libraries

- numpy
- scipy
- sklearn “scikit-learn”

These are installed on studentweb.cmix.louisiana.edu

Arrays in Python

```
Python 2.7.14 (default, Oct 16 2017, 00:13:25)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> import scipy
>>> arr = np.array([[1,2,3],[4,5,6]])
>>> arr
array([[1, 2, 3],
       [4, 5, 6]])
>>> arr.shape
(2, 3)
>>> type(arr)
<type 'numpy.ndarray'>
>>> arr.dtype
dtype('int64')
```

```
>>> arr
array([[1, 2, 3],
       [4, 5, 6]])
>>> arr[0,1]
2
>>> arr[0,:]
array([1, 2, 3])
>>> r1=arr[1,:]
>>> c2=arr[:,2]
>>> r1
array([4, 5, 6])
>>> c2
array([3, 6])
>>>
```

Matrix Operations

```
>>> np.transpose(arr)
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> bvec=np.transpose(np.array([5,0,9]))
>>> bvec
array([5, 0, 9])
>>> bvec.shape
(3,)
>>> np.matmul(arr, bvec)
array([32, 74])
>>> arr
array([[1, 2, 3],
       [4, 5, 6]])
>>> bvec
array([5, 0, 9])
```

```
>>> cvec=np.array([5,0,9])
>>> cvec.shape
(3,)
>>> np.matmul(arr,cvec)
array([32, 74])
```

Let's Generate Some Data

```
>>> np.random.uniform(0.1, 1.5)
0.475697472392474
>>> [np.random.uniform(0.1, 1.5) for i in xrange(20)]
[0.24330739601121412, 0.771935852518417l, 1.402958469973410 3,
0.26020160176488016, 0.20474323017557178, 0.778387846318867 8,
0.734039900708376, 0.4957795919454316 4, 0.860250644654592 6,
1.1332077476379507, 0.877349114229437 8, 1.287352602354815 6,
0.7223653990799094, 0.830713721797704 4, 0.159195487195337 38,
1.1080242723567986, 1.356748774814495, 1.365686607869794 5,
1.3335391689388925, 1.1128240051250644]
```

Let’s Generate Some 2D Data

```
>>> xt = [[np.random.uniform(0.1, 1.5) for i in xrange(20)],
... [np.random.uniform(-1.0, 1.0) for i in xrange(20)]]
>>> type(xt)
<type 'list'>
>>> xt.shape
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'list' object has no attribute 'shape’
>>> xt=np.array(xt)
>>> xt.shape
(2, 20)
>>> type(xt)
<type 'numpy.ndarray'>
```

```
>>> xt
array([[ 0.2658256 ,  0.44786858,  0.82669208,  1.41763826,  0.31273601,
         0.50613847,  1.42664273,  0.12457044,  0.14026878,  0.57028987,
         0.80109158,  0.12333076,  0.97052285,  1.03812377,  0.93374927,
         1.37707205,  0.88549127,  0.70678135,  0.83217132,
0.60713106],
       [ 0.56541641, -0.28761058, -0.92887069,  0.0990287 , -0.4403601 ,
        -0.78906364,  0.83732901, -0.17800827, -0.69459559,  0.1341467 ,
        -0.9640439 , -0.32261052, -0.85519285,  0.99936408, -0.37007427,
        -0.21879445, -0.63226485, -0.39706442,  0.17331681,
0.09818469]])
>>> xdata = np.transpose(xt)
>>> xdata.shape
(20, 2)
>>> xdata
array([[ 0.2658256 ,  0.56541641],
       [ 0.44786858, -0.28761058],
       [ 0.82669208, -0.92887069],
       [ 1.41763826,  0.0990287 ],
       [ 0.31273601, -0.4403601 ],
…
```

Targets

```
>>> tvec = np.ones(20)
>>> type(tvec)
<type 'numpy.ndarray'>
>>> tvec
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

Generate Class-0 Data

```
>>> x0data = np.transpose(np.array([[np.random.uniform(-1.5, -0.1) for i
in xrange(20)],
... [np.random.uniform(-1.0, 1.0) for i in xrange(20)]]))
>>> x0data.shape
(20, 2)
>>> t0vec = -np.ones(20)
```

Both Classes Now

```
>>> x1data = xdata
>>> t1vec = tvec
>>> xdata = np.concatenate((x1data, x0data), axis=0)
>>> xdata.shape
(40, 2)
>>> tvec = np.concatenate((t1vec, t0vec))
>>> tvec.shape
(40,)
```

Both Classes Now

```
>>> shuffle_index = np.random.permutation(40)
>>> xdata, tvec = xdata[shuffle_index], tvec[shuffle_index]
>>> tvec
array([ 1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,
       -1., -1.,  1.,  1.,  1., -1., -1.,  1.,  1., -1.,  1., -1.,  1.,
       -1., -1., -1.,  1., -1., -1., -1.,  1., -1., -1., -1., -1., -1.,
-1.])
>>> xdata
array([[ 0.97052285, -0.85519285],
       [ 0.50613847, -0.78906364],
       [ 1.37707205, -0.21879445],
       [ 0.44786858, -0.28761058],
       [ 1.41763826,  0.0990287 ],
       [-0.82782959, -0.71167813],
…
```

Classifier

```
>>> import sklearn
>>> from sklearn.linear_model import Perceptron
>>> classifier = Perceptron(tol=1e-3, random_state=0)
>>> classifier.fit(xdata, tvec)
Perceptron(alpha=0.0001, class_weight=None, early_stopping=False,
eta0=1.0,
      fit_intercept=True, max_iter=None, n_iter=None,
n_iter_no_change=5,
      n_jobs=None, penalty=None, random_state=0, shuffle=True,
tol=0.001,
      validation_fraction=0.1, verbose=0, warm_start=False)
>>> classifier.coef_
array([[ 1.3923561 ,  0.15753578]])
>>> classifier.intercept_
array([ 0.])
>>> classifier.n_iter_
7
```

After the "classifier.fit" call, the Perceptron algorithm learns the dividing plane, specified by "coef_" and "intercept_" parameters. The number of iterations it took to converge (or stop) is given by n_iter_

Classifier

```
>>> import sklearn
>>> from sklearn.linear_model import Perceptron
>>> classifier = Perceptron(tol=1e-3, random_state=0)
>>> classifier.fit(xdata, tvec)
Perceptron(alpha=0.0001, class_weight=None, early_stopping=False,
eta0=1.0,
      fit_intercept=True, max_iter=None, n_iter=None,
n_iter_no_change=5,
      n_jobs=None, penalty=None, random_state=0, shuffle=True,
tol=0.001,
      validation_fraction=0.1, verbose=0, warm_start=False)
>>> classifier.coef_
array([[ 1.3923561 ,  0.15753578]])
>>> classifier.intercept_
array([ 0.])
>>> classifier.n_iter_
7
```

Is the solution correct?

Is the Solution Correct?

Class-1 samples have positive x_0 values while Class-0 samples have negative x_0 values

The ideal solution is therefore $x_0 = 0$

The perceptron algorithm learned the solution given by

```
>>> classifier.coef_
array([[ 1.3923561 ,  0.15753578]])
>>> classifier.intercept_
array([ 0.])
```

The solution is therefore: $1.39\, x_0 + 0.16\, x_1 + 0 = 0.$
Or $x_0 + 0.12\, x_1 = 0$, which is reasonably close to the ideal solution.

Is the Solution Correct?

```
>>> np.equal(classifier.predict(xdata), tvec)
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True], dtype=bool)
```

classifier.predict(xdata) produces the output of using the learned solution to classify the training data xdata

We see from above that the training samples are all correctly classified.

Make the Data Not Linearly Separable

Let us change two of the target values so that the training data set is no longer linearly separable; i.e. no line can separate the two classes of samples

```
>>> tbad = tvec
>>> tbad
array([-1., -1., -1.,  1.,  1.,  1., -1., -1.,  1.,  1.,  1., -1., -1.,
        1., -1., -1., -1.,  1., -1.,  1., -1.,  1., -1.,  1., -1., -1.,
       -1.,  1.,  1., -1., -1.,  1.,  1.,  1.,  1., -1.,  1., -1.,  1.,
1.])
>>> tbad[0] = 1
>>> tbad[3] = -1
```

As a side note, “tbad = tvec” does not make a copy of tvec; the array tvec will be modified by the above operation. I change the variable name so that it is obvious that we are dealing with a different target list

Train a Different Classifier

```
>>> cbad = Perceptron(tol=1e-3, random_state=0)
>>> cbad.fit(xdata, tbad)
Perceptron(alpha=0.0001, class_weight=None, early_stopping=False,
eta0=1.0,
      fit_intercept=True, max_iter=None, n_iter=None,
n_iter_no_change=5,
      n_jobs=None, penalty=None, random_state=0, shuffle=True,
tol=0.001,
      validation_fraction=0.1, verbose=0, warm_start=False)
>>> cbad.coef_
array([[ 1.44547412, -1.35469276]])
>>> cbad.n_iter_
15
>>> cbad.intercept_
array([ 0.])
```

Different parameters because the training samples have been modified

Perceptron Cannot Find a Line to Separate Two Classes that Are Not Linearly Separable

```
>>> cbad.predict(xdata)
array([-1.,  1., -1., -1.,  1.,  1., -1.,  1.,  1.,  1.,  1., -1., -1.,
        1., -1., -1., -1.,  1., -1.,  1., -1.,  1., -1., -1., -1., -1.,
        1.,  1.,  1.,  1., -1.,  1., -1.,  1.,  1., -1., -1., -1.,  1.,
1.])
>>> tbad
array([ 1., -1., -1., -1.,  1.,  1., -1., -1.,  1.,  1.,  1., -1., -1.,
        1., -1., -1., -1.,  1., -1.,  1., -1.,  1., -1.,  1., -1., -1.,
       -1.,  1.,  1., -1., -1.,  1.,  1.,  1.,  1., -1.,  1., -1.,  1.,
1.])
>>> np.equal(cbad.predict(xdata), tbad)
array([False, False,  True,  True,  True,  True,  True, False,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False,  True,  True, False,
        True,  True, False,  True,  True, False,  True,  True,  True,
       False,  True,  True,  True], dtype=bool)
```

Note that additional errors are made, beyond the two points that were “flipped” from one class to the other