Week 1: Programming Exercise

Perceptron Vs. Linear Regression for Classification

In this exercise we will compare the Perceptron learning algorithm with the linear regression model for doing classification. Along the way, we will get some acquaintance with Python, and particular NumPy, as a language for machine learning. In this exercise the (three) data sets have the following form, $D = \{(x_i, y_i) \mid x_i \in \mathbb{R}^2, y \in \{-1, 1\}\}$, that is, the input domain for the point sets is two dimensional which allows us to visualize them. In the data sets we have already added the dummy variable $x[0] = 1$ for all the points. So do not let the three dimensions fool you, and plot only the relevant two.

The perceptron algorithm finds a hyperplane represented by a vector w_{pla} that splits the input domain such that all points with $+1$ labels is on one side of the hyperplane and the points with labels -1 on the other (if that is possible). Linear regression computes the linear model (line in 2D) that best approximates the real valued target (we compute the vector w_{lin} minimizing $\sum_i (w_{\text{lin}}^\intercal x_i - y_i)^2$). The labels are either -1 or $+1$, but that is of course a small subset of the real numbers, so linear regression finds the best linear fit to these values over the input points.

Steps to the Goal Line

Load the data and plot it. Nice to have a visualization to start with. Consider plotting the labelled sets $(X^+, X^-) = (\{x \in D \mid y = +1\}, \{x \in D \mid y = -1\})$ with separate colors and markers. Note that for all data sets the data is linear separable so the perceptron learning algorithm will finish. Figure out how to plot a hyperplane (w). Note that w has three parameters w_0, w_1, w_2 and you want to plot the line $\sum_{i=0}^{2} w_i x_i = w^\intercal x = 0$ in 2D ($x_0 = 1$ still does not need plotting).

Implement Perceptron and Linear Regression Learning Algorithms Follow the description in the book or from the slides. You may want to print how many iterations the Perceptron algorithm needs.

Apply Learning Algorithms and Visualize Results For both data sets, run the learning algorithms and compute a set of parameters for each model. Compute the in sample error for both models and print to screen ($\sum_i \text{er}(h(x_i), y_i)$, where $\text{er}(x, y)$ is zero if $x = y$ and one otherwise). Plot the decision boundaries on the plot with the input points to compare the hyperplanes you found visually. Can you explain the results? Please try!

Using Python, NumPy and Matplotlib

Use Python preferably Python 3, but 2.7 is also reasonable. Interactive Python or IPython is good way to develop and test code. Your code should begin with the two imports:

```
import numpy as np
import matplotlib.pyplot as plt
```

Load and Save Data Use the NumPy commands load and savez.

```
dat = np.ones((7, 3))
target = -1 * np.ones((7,))
np.savez('myfile.npz', dat=dat, target=target)
stuff = np.load('myfile.npz')
dat = stuff['dat']
target = stuff['target']
```

Arrays

```
a = np.array([1, 2])
print(a.ndim)  # 1
a = np.array([[1, 2], [2, 3]])
print(a.ndim)  # 2
a = np.array([1, 2, 3])
print(a.reshape((-1, 1)))
# array([[1],
#        [2],
#        [3]]) (column vector)
print(a.reshape((1, -1)))  # array([1, 2, 3]) (row vector)
print(np.ones((2, 2)))
# array([[ 1.,  1.],
#        [ 1.,  1.]])
m = np.ones((5,3))
m[0,0] = 2
m[4,2] = 3
m[0,:] # first row [2,1,1]
m[:,-1] # last column [1,1,1,1,3]
```

Pointwise operators vs. matrix multiplication In NumPy, $a * b$ means pointwise multiplication. To get matrix multiplication or inner product use `np.dot`. Note that `a.ndim` must be equal to `b.ndim`. If they are not, use `reshape`.

```
a = np.array([1, 2])
b = np.array([2, 3])
print(a * b) # array([2, 6])
print(np.dot(a, b)) # 8
```

Matplotlib for visulization Simple plotting is done as follows.

```
dat = np.arange(1, 10)
plt.plot(dat, 2*dat, 'r-s', linewidth=2, markersize=5)
plt.show()
```

The commands `hold`, `draw`, `xlim`, `ylim` may come in handy as well.

Boolean arrays (or filters) If you write `pos = (y == 1)`, you will get an array `pos` with booleans (`True` and `False`). An array of booleans is sometimes called a filter. A filter can be used in indexing, e.g. `X[pos]` to retrieve the data points x_i for which $y_i = 1$. To get the `True`-entries of a filter, write `indices = pos.nonzero()[0]`. For more info, see the NumPy documentation for `nonzero`.

If a and b are two NumPy filters, you can combine them with Python's bitwise operators: `a & b` is the intersection, `a | b` is the union, and `~a` is the complement of a. This is useful when plotting e.g. the data points with label -1 that are classified incorrectly by the hyperplane, or combined with `np.random.choice(indices)` to sample a random incorrectly classified data point.

IPython is your friend You can load and reload code from code files and use in IPython. The hardest part is reloading so here it goes.

```
from importlib import reload  # Python 3 specific
import mycode
mycode.main()
reload(mycode)
mycode.main()
```

`https://docs.python.org/3/library/importlib.html#importlib.reload`
If you forget your variables type `whos`.