Week 2: Programming Exercise

Nonlinear Transforms

In this exercise you will experiment with nonlinear transforms for adding expressive power to our liner models. As in the lectures the assignment considers adding polynomial features. The input data is in two dimensions so you can visualize the results. Note that this time we have not added the dummy column of all ones into the feature data matrix. We denote a feature transform by ϕ. In the lectures we showed you a simple non linear transform that was able to separate a simple circular data set in feature space. This was the transform $\phi(x_1, x_2) = (x_1^2, x_2^2)$. You are going to need the code for last weeks coding exercise so if you have not made that exercise it is a good idea to do so first.

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. If you did last weeks excercise you should already know how to do this.

Implement Perceptron Learning Algorithm If you already did this last week you are done.

Base Case: Transform, Learn and Visualize To have something to start with you should use the feature transform from the lecture (described above) on the first data set and run the perceptron learning algorithm on the transformed data. If you do the transform correctly the data will be become linear separable in the feature space and the perceptron finds a separating hyperplane.

Now you need to visualize the decision boundary in the original input space, e.g. you need ϕ^{-1} of the decision boundary in the transformed space. For that we suggest using contour plots which are described briefly in the code help section below.

General Case: The remaining data sets are not linear separable with the simple transform we have seen so far. You can alter the perceptron to run for T iterations and returning the best hyperplane found (the hyperplane minimizing in sample error) and see how well you can do. Another approach which you must try is to come up with a better non-linear transform that makes the data sets linear separable in the transformed space. We suggest (wink wink) you use the general form of polynomials of degree 3, e.g:

$$\phi(x_1, x_2) = \left(x_1^i x_2^j \quad \text{for} \quad 0 \leqslant i + j \leqslant 3 \right)$$

This should give you $\binom{2+3}{3} = 10$ features where one of them is the all ones vector.

Using Python, NumPy and Matplotlib

Contour Plots are way of visualizing weighed point sets using colors and contour lines that show how the point set weights vary across the the input domain. For instance you could weigh an input set with the result of applying the nonlinear transform and applying the learned hypothesis in the transformed space.

```
xs = ys = np.linspace(-1, 1, 100)
y, x = np.meshgrid(ys, xs)
z = x^2 - y
plt.contour(xs, ys, z, [0])
```

Shape of an array One often needs to know the size of a vector (for instance the size of the input data matrix X).

```
a = np.ones((24, 42))
print(a.shape)  # (24, 42)
print(a.shape[0])  # 24
print(a.shape[1])  # 42
```

keepdims keepdims is an option we can give to most functions in NumPy so we do not change the number of dimensions when we apply aggregating functions such as min, max or sum. For instance, we can compute the sum of each row in a matrix, and without keepdims, that reduces the number of dimensions to just one, so the result is no longer two dimensional.

```
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a.shape)  # (2, 3)
print(np.sum(a))  # 15
b = np.sum(a, axis=0)  # [5, 7, 9] (summing columns)
c = np.sum(a, axis=1)  # [6, 15] (summing rows)
print(b.shape)  # (3,) (axis 0 disappeared)
print(c.shape)  # (2,) (axis 1 disappeared)
d = np.sum(a, axis=0, keepdims=True)
e = np.sum(a, axis=1, keepdims=True)
print(d.shape)  # (1, 3) (axis 0 smashed to length 1)
print(e.shape)  # (2, 1) (axis 1 smashed to length 1)
```

Notice how the number of dimensions of b and c are one less than a, and notice how d, e does not have this problem because we told the sum algorithm not to shave away a dimension. In this setup, $a + b$ is an error because the number of dimensions is different, but $a + d$ works because of the NumPy feature of *broadcasting*.