Machine Learning *COMP 5/6630*

HW05: Kernel methods

Remember that only PDF submissions are accepted.

1. Suppose that you have N data points in D dimensions. Suppose you run perceptron for 1 pass over the data set and it makes K updates. How long does this take? (Big-O notation, please: note, it should not actually depend on K.) How long would it take to run if you preprocessed your data with the quadratic feature map? How long for cubic feature map?

   Answer: O(N.D)

   O(N.D^2)

   O(N.D^3)

   Now, suppose that you run kernelized perceptron over the same data with a linear kernel. How long will this take (it should depend on K now, and note that under different feature maps, the numbers K will not be comparable.) What about for quadratic or cubic kernels?

   Answer: For linear -> O(N.D)

   For Quadratic and Cubic -> O(N^2.D)

2. (Programming) you will extend the perceptron training and prediction to use polynomial kernels. A polynomial kernel of degree d is defined as:

   $$K^d_{poly}(x, z) = (a + bx^T z)^d$$

   Here, $a \geq 0$, $b > 0$ and $d \in \{1,2,...,\infty\}$ are hyperparameters of the kernel.

   Just hand-in your code, and you don't have to play with data for now. (But you are encouraged to play with some toy data you find on your own to validate if your implementation is correct or not). You can reuse the code to this question in your mini-project, and play with data then. See next page!

```python
import numpy as np

def polynomial_kernel(x, y, p=3):
    return np.power((1+np.dot(x,y)), p)

class KernelPerceptron(object):

    def __init__(self, kernel=polynomial_kernel, T=1):
        self.kernel = kernel
        self.T = T

    def fit(self, X, y):
        n_samples, n_features = X.shape
        #initializing zero array
        self.alpha = np.zeros(n_samples, dtype=np.float64)

        # Gram matrix
        K = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                K[i,j] = self.kernel(X[i], X[j])

        for t in range(self.T):
            for i in range(n_samples):
                if np.sign(np.sum(K[:,i] * self.alpha * y)) != y[i]:
                    self.alpha[i] += 1.0

    # predict functions:
    def predict(self, X):
        result = 0
        for i in range(len(X)):
                result += self.alpha[i] * self.y[i] * self.kernel(self.X, self.X[i])
        return 1 if result > 0 else -1
```