582669 Supervised Machine Learning (Spring 2011)
Homework 4

Turn this homework in no later than **Tuesday, 15 February, at 15:00**. As usual, there are four problems.

1. We generalise the Perceptron Algorithm by introducing a learning rate $\eta > 0$. The update becomes

$$w_{t+1} = w_t + \eta \sigma_t y_t x_t.$$ 

Further, we start the algorithm with $w_1 = w_{\text{init}}$ where the initial weights need not be zero. (Note that if we have $w_{\text{init}} = 0$ then the learning rate does not affect the predictions $\text{sign}(w_t \cdot x_t)$.)

Assume that $||x_t||_2 \leq X$ for some $X > 0$, and some $u \in \mathbb{R}^d$ satisfies $y_t u \cdot x_t \geq 1$ for all $t$. Modify the proof for the Perceptron Convergence Theorem by using

$$P_t = \frac{1}{2}||u - w_t||_2^2$$

as the potential function. The result should be that

$$\sum_{t=1}^{T} \sigma_t \leq ||u - w_{\text{init}}||_2^2 X^2$$

for a suitable choice of $\eta$. Thus, if we start the algorithm close to the target, we get a smaller mistake bound.

*Hint:* This is a fairly straightforward modification of the proof in the lecture notes. Instead of $c$ and $\gamma$, the learning rate $\eta$ will appear in some terms of the potential estimate.

2. As with the all subsets kernel (Example 2.19, page 105), define for $A \subseteq \{1, \ldots, n\}$ the feature

$$\psi_A(x) = \prod_{i \in A} x_i.$$ 

The degree $q$ ANOVA feature map has the $\binom{n}{q}$ features $\psi_A$ where $|A| = q$. (Thus the all subsets feature map combines the ANOVA features for $q = 0, \ldots, n$.)

Let $k_q$ be the kernel of this feature map. There is no nice closed form for this kernel, but given $x, z \in \mathbb{R}^n$ we can still compute the value

$$k_q(x, z) = \sum_{|A|=q} \psi_A(x) \psi_A(z)$$

much more efficiently than the naive $O(n^q)$. Give an algorithm to do this.

*Hint:* Express $k_q((x_1, \ldots, x_n), (z_1, \ldots, z_n))$ in terms of $k_{q-1}((x_1, \ldots, x_{n-1}), (z_1, \ldots, z_{n-1}))$ and $k_q((x_1, \ldots, x_{n-1}), (z_1, \ldots, z_{n-1}))$. You can save computation effort by dynamic programming.
3. Consider online linear regression, where now $\hat{y}_t$ and $y_t$ can both be arbitrary real numbers. The analogue of the Perceptron algorithm is the Least Mean Squares algorithm (LMS, also known as Widrow-Hoff):

Initialise $w_1 = 0$. 
Repeat for $t = 1, \ldots, T$: 
1. Get $x_t \in \mathbb{R}^n$.
2. Predict $\hat{y}_t = w_t \cdot x_t$.
3. Receive the correct answer $y_t$.
4. Update $w_{t+1} = w_t - \eta(y_t - \hat{y}_t)x_t$.

Here $\eta > 0$ is a learning rate parameter.

Assume that there are some $u \in \mathbb{R}^n$ and $X > 0$ such that $y_t = u \cdot x_t$ and $||x_t||_2 \leq X$ for all $t$. Show that the square loss of the LMS algorithm can be bounded as

$$\sum_{t=1}^{T} (y_t - \hat{y}_t)^2 \leq ||u||^2_2 X^2.$$

**For extra credit** (worth one regular problem), generalise this to the “agnostic” case where we do not assume $u \cdot x_t = y_t$.

**Hint:** For the basic case, show that

$$\frac{1}{2}||u - w_t||^2_2 - \frac{1}{2}||u - w_{t+1}||^2_2 = (\eta - \frac{1}{2}\eta X^2) (y_t - \hat{y}_t)^2.$$

Optimise $\eta$ and sum over $t$.

For the agnostic case, show that

$$\frac{1}{2}||u - w_t||^2_2 - \frac{1}{2}||u - w_{t+1}||^2_2 = a(y_t - \hat{y}_t)^2 - b(y_t - u \cdot x_t)^2$$

for some $a, b > 0$ that depend on $X$ and $\eta$. You do not need to find the optimal $\eta$ for this case.
4. Consider binary classification with the instance space \( X = [-1,1]^n \). Choose \( n = 10 \) and, as target, the second-order polynomial classifier \( f(x) = \text{sign}(g(x)) \) where \( g(x) = x_1^2 + x_1 x_2 - x_2^2 \). Generate a sample of data points (a few hundred should be enough) \((x_t, y_t)\) where \( x_t \) are drawn uniformly at random from \( X \), and \( y_t = f(x_t) \). To enforce a positive margin, drop all points where \(|g(x)| \leq 0.2\). Split your data into two equally large sets, a training set and a test set.

Implement a kernel version of the marginalised perceptron algorithm. In other words, combine the kernel trick (Algorithm 2.15) with the Marginalised Perceptron (Algorithm 2.21). As the kernel, use \( k(x, z) = (x \cdot z)^2 \).

First run your algorithm on the training data with \( \rho = 0 \) until it converges, i.e., you obtain a consistent classifier. Then test your classifier on the test set. Try different sizes of training set; how large a training set do you need to make your classifier perform well also on the test data?

Now try positive values of \( \rho \). How large a margin can you achieve on the training set? Does going for a large margin on the training set affect the performance on test data if the training set is small?

Your solution should consist of a brief explanation of the observations you made, perhaps some representative figures to illustrate this, and a printout of your program code.

Implementation issues  Making the algorithm converge can take several iterations through the whole training set (and quite a few to get a good margin). You also need to experiment a little to find a good learning rate \( \eta \). A reasonable starting point might be \( \eta = 0.1 \). Don’t spend excessive time trying to get the best possible result, but you should get a clearly positive margin.

Keep your input vectors \( x_t \) in a fixed table. Since you will iterate through the data several times, it may save time to pre-compute all the values \( k(x_t, x_j) \) into an \( m \times m \) matrix (often called the Gram matrix, or just the kernel matrix). What your algorithm updates is just the coefficients \( \alpha_i \) in the kernel expansion \( \sum_i \alpha_i k(\cdot, x_i) \). For the normalisation step, notice that you can write \( ||w||^2 = \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j) \).