CS475 Machine Learning, Fall 2012: Homework 4

Dan Crankshaw

Question 1:

(a) We should use the first model, the one with less support vectors. The less support vectors we have, the less variance in our model. The first model will have less overfitting issues than the second model.

(b) Not necessarily. The dual and primal forms should be functionally equivalent to each other, and thus should learn the same model on the same data. The amount of overfitting in the dual form depends on your choice of kernel (see question 3). The advantages of the dual form are that if $m >> n$ than it can be computationally more efficient, and it allows for non-linear kernels using the kernel trick.

Question 2:

(a) To show that $h(a) = max(1 - a, 0)$ is convex, we must show that it satisfies the condition $h(tx_1 + (1-t)x_2) \leq th(x_1) + (1-t)h(x_2)$ for $t\epsilon[0, 1]$ and all x_1, x_2. The left hand side (LHS) of this condition when plugged into the function is equivalent to $max(1 - x_2 + t(x_2 - x_1), 0)$. Without loss of generality, we can assume that $x_1 < x_2$. We consider three cases.

i) $x_1 > 1$ and $x_2 > 1$
LHS: $max(1 - x_2 - t(x_1 - x_2), 0)$
RHS: 0
We know that $1 - x_2 < 0$ and because $x_1 > 1$ and $t < 1$, $|1 - x_2| > |x_1 - x_2|$ and so the t term will not make the expression greater than 0. Thus, the LHS will always evaluate to 0. The RHS is also 0, so our condition is fulfilled for this case.

ii) $x_1 < 1$ and $x_2 > 1$
LHS: $max(1 - x_2 - t(x_1 - x_2), 0)$
RHS: $t(1 - x_1)$
$1 - x_2 < 0$ like in the previous case, and $x_2 > 1$. This means that $t(x_2 - x_1) < t(1 - x_1)$. Let $a = 1(1 - x_1)$, and let *epsilon* be some positive real number. Thus, we have

$$max(1 - x_2 + (a - \epsilon), 0) < a \tag{1}$$

satisfying our condition for this case.

iii) $x_1 < x_2 < 1$
LHS: $max(1 - x_2 - t(x_1 - x_2), 0)$
RHS: $t(1 - x_1) + (1 - t)(1 - x_2)$
The right hand side simplifies to $r = 1 - x_2 + t(x_2 - x_1)$. Thus our condition becomes $max(r, 0) \leq r$. If we can show that $r >= 0$, then we our condition will be satisfied. Notice that $1 - x_2 > 0$ because either x_2 is negative (turning the difference into a sum) or $x_2 < 1$. Also notice that $t(x_2 - x_1) > 0$ because $x_2 - x_1$ if $x_2 > 0$, then x_1 is either a smaller positive number or a negative number turning the difference into the sum of two positive numbers. And if $x_2 < 0$ then $|x_2| < |x_1|$ and so the we have a smaller negative number minus a bigger negative number, yielding a positive value. Thus, r is always the sum of two positive numbers, and thus $r > 0$, satisfying our condition.

Therefore, we have satisfied the condition for $H(a)$ to be convex.

(b) The disadvantage of this function is that it does not allow for a margin around the separating hyperplane. Instead of allowing us to have a constraint that the closest point to the separating hyperplane w be 1, it allows the distance to be zero. This makes it impossible to find the max margin separation with this loss function, because as soon as a point is on the correct side of the decision boundary it will have zero loss, instead of as soon as it is outside of the margin. This cannot be fixed with a scaling factor.

(c) This question asks how we can scale λ so that learning a decision boundary with a scaled minimum margin of 0.5 is equivalent to our original formulation where our scaled minimum margin was 1. In the case where the margin is 0.5, our scaling on w is 0.5 times the original scaling on w. Therefore, if we use both our modified loss function but also enforce that the scaling on w is half the original scaling, we should get the same result as the original formulation. Because the $||w||^2$ margin term is quadratic, we get a $.5^2$ in front of $||w||^2$, so $\lambda' = 0.25\lambda$.

Question 3:

(a) Increasing d in a polynomial kernel will make over-fitting more likely. Increasing d amplifies the small differences in similarities between different pairs of examples. The higher the relative kernel values for an example are, the higher the learned α will be, and that example will be a support vector. By amplifying potentially small differences in kernel values, the model becomes much more sensitive to small changes in the training data. Thus, it over-fitting is more likely to occur.

(b) Increasing σ has the opposite effect as the previous problem. Increasing σ decreases the relative values of the kernels, and decreases the amount small differences in examples change the model, leading to less over-fitting.

(c)

$$K(x, x') = K_1(x, x') + K_2(x, x') \tag{2}$$

means that there must exist some Φ such that

$$< \Phi(x), \Phi(x') > = < \Phi_1(x), \Phi_1(x') > + < \Phi_2(x), \Phi_2(x') > \tag{3}$$

This condition is easily satisfied by picking as our basis function Φ the concatenation of Φ_1 and Φ_2. Thus,

$$< \Phi(x), \Phi(x') > = < (\Phi_1(x)\Phi_2(x)), (\Phi_1(x')\Phi_2(x') > \tag{4}$$

$$= < \Phi_1(x), \Phi_1(x') > + < \Phi_2(x), \Phi_2(x') > \tag{5}$$

Question 4:

(a) The computational complexity of prediction of a linear SVM in the primal form is $O(m)$. It is just a function of $w^T x$, and independent of the number of training examples at prediction time.

(b) The computational complexity of prediction of a non-linear SVM will be $O(sm)$. We are summing over the support vectors s. And for each support vector x_i we compute $\alpha_i K(x, x_i)$. Computing the kernel $K(x, x_i)$ once is $O(m)$. Thus the complexity is $O(sm)$.

Question 5:

(a) We compute the gradient of β over some subset k of the examples.

$$\frac{\partial}{\partial \beta_j} = \frac{1}{k} \sum_{i=1}^{k} 2(y_i - x_i^T \beta)(-x_{ij}) + 2\lambda\beta_j \tag{6}$$

We have m partial derivatives in the gradient, and computing each partial is $O(km)$, so the total complexity is $O(km^2)$.

(b) Increasing k just increases the computational complexity linearly. This results in slower updates. However, increasing k looks at more data at once. If the data is similar, this brings little advantage. But if the data is relatively heterogeneous or noisy then this will make the gradient update better.

(c) Neural networks (can) use stochastic gradient descent. In neural networks, we use gradient ascent to maximize our objective function. We can use stochastic gradient ascent to compute these gradients more efficiently.