CS 475 Machine Learning: Homework 3
Supervised Classifiers 2

Due: Friday October 16, 2015, 11:59pm

100 Points Total Version 1.0

Make sure to read from start to finish before beginning the assignment.

1 Analytical (50 points)

The following problems consider a standard binary classification setting: we are given n observations with m features, $x_1, ..., x_n \in \mathbb{R}^m$.

1) Overfitting (8 points) SVMs using nonlinear kernels usually have two tuning parameters (regularization parameter C and kernel parameter γ), which are usually determined by cross validation.

(a) Suppose we use cross validation to determine the non-linear kernel parameter and slack variable for an SVM. We find that classifiers using parameters (c_1, γ_1) and (c_2, γ_2) achieve the same cross validation error, but (c_1, γ_1) leads to fewer support vectors than (c_2, γ_2). Explain which set of parameters should we choose for the final model?

Answer:
We should choose the set (c_1, γ_i) because fewer support vectors means that there are less trained samples on the margin of the hyperplane (boundary) for the support vector machine. Less samples on the margin means that the hyperplane is more general for other test samples. Say, in the worst case, that all the sample points are in the margin. It is best to have less support vectors because we want the boundary to have more confidence on which side of the boundary a sample belongs. We also want the margin between the closest support vector mechine and the boundary to maximized.

(b) The optimization problem of linear SVMs can be in either primal or dual form. As we know, the primal form has m parameters to learn, while the dual form has n parameters to learn. If $m \gg n$, is it true that the dual form reduces over-fitting since it has fewer parameters? Explain.

Answer:
It should not reduce overfitting because it produces the same hyperplane. It's just a different approach to the same hyperplane.

2) Hinge Loss (12 points) Linear SVMs can be formulated in an unconstrained optimization problem

$$\min_{w,b} \sum_{i=1}^{n} H(y_i(w^T x_i)) + \lambda \|w\|_2^2, \tag{1}$$

where λ is the regularization parameter and $H(a) = \max(1-a, 0)$ is the well known hinge loss function. The hinge loss function can be viewed as a convex surrogate of the 0/1 loss function $I(a \leq 0)$.

(a) Prove that $H(a)$ is a convex function of a.

To show something is convex we need to show that

$$\begin{aligned} \lambda h(x_1) + (1-\lambda)h(x_2) &\geq h(\lambda x_1 + (1-\lambda)x_2) \\ \lambda max(1-x_1, 0) + (1-\lambda)max(1-x_2, 0) &\geq max(1-(\lambda x_1 + (1-\lambda)x_2), 0) \\ &\geq max(1 - \lambda x_1 - x_2 + \lambda x_2, 0) \\ &\geq max(1 - x_2 + \lambda(x_2 - x_1), 0) \end{aligned}$$

Now we have four cases

(a) $x_1 < 1$ and $x_2 < 1$
Because both x_i are less than 1, we have max(1 - x_i, 0) will always turn out to be 1-x_i. We also know that the RHS will be the $1-a$ value where $a = 1 - x_2 + \lambda(x_2 - x_1)$. This is because

$$\begin{aligned} \lambda(1-x_1) + (1-\lambda)(1-x_2) &\geq 1 - x_2 + \lambda(x_2 - x_1) \\ \lambda - \lambda x_1 + 1 - x_2 - \lambda + \lambda x_2 &\geq \\ 1 - x_2 - \lambda x_2 + \lambda x_2 &\geq \\ 1 - x_2 + \lambda(x_2 - x_1) &\geq \end{aligned}$$

which is always true because LHS = RHS.

(b) $x_1 < 1$ and $x_2 \not< 1$
We know the RHS evaluates to 0 because $x_2 > 1$ and thus the equation evaluates to $max(-num, 0)$ which will always be 0.

$$\begin{aligned} \lambda(1-x_1) &\geq 0 \\ \lambda - \lambda x_1 &\geq 0 \end{aligned}$$

We know this is always true because $\lambda \in [0, 1]$ and thus $\lambda > \lambda x_1$ because $x_1 < 1$.

(c) $x_1 \not< 1$ and $x_2 < 1$ RHS will always be 0 because $x_1 > x_2$ and thus we are always subtracting from 1 and will thus evaluate to $max(-num, 0)$ again.

$$\begin{aligned}
(1-\lambda)(1-x_2) &\geq 0\\
1-\lambda-x_2+\lambda x_2 &\geq 0\\
1-x_2-\lambda(1-x_2)+\lambda x_2 & geq0\\
\lambda((1-x_2)-(1-x_2)) &\geq 0\\
0 &\geq 0
\end{aligned}$$

(d) $x_1 \not< 1$ and $x_2 \not< 1$ Both the LHS and RHS evaluate to 0 because both the max's evaluate to 0 and the RHS has an $1-x_2$ where $x_2 > 1$.

$$0 \geq 0$$

Because the equation is true in all cases, we can show that the function is convex.

(b) The function $L(a) = \max(-a, 0)$ can also approximate the 0/1 loss function. What is the disadvantage of using this function instead?

Answer: The disadvantage to using this function instead of the hinge loss function lies in the margin. In the H(x) function, we penalize the value when we have a point that lies with in the margin, even if it is correctly classified. However, if we have that L(x) function, then it does not matter if the sample point lies in the margin, we just count it as correct. We do not change our margin based on the number of points that lie in it. Thus, we do not find a good boundary where the hyperplane has a large margin between the nearest points. It is no better than a basic perceptron.

(c) If $H'(a) = \max(0.5 - a, 0)$, show that there exists λ' such that (2) is equivalent to (1). Hint: think about the geometric interpretation of hinge loss.

$$\min_{w,b} \sum_{i=1}^{n} H'(y_i(w^T x_i)) + \lambda' \|w\|_2^2. \tag{2}$$

Answer: If we look at the geometric interpretations of the hinge loss, we see that there is a linear decreasing in the loss until we get to $y_i(w^T x_i) = 0.5$, where we plateau at zero (i.e., for any value of $y_i(w^T x_i) > 0.5$ we get loss as 0, but otherwise we take some penalty). This means we take a penalty for any support vectors being within half of the margin. To compensate (to get it look like of the graph of $H(a) = max(1-a, 0)$) we can conditionally set $\lambda' =$

$$\begin{aligned}
\lambda + \frac{0.5}{\|w\|_2^2} &\text{ if } H'(a) > 0\\
\lambda + \frac{max(1-a,0)}{\|w\|_2^2} &\text{ otherwise}
\end{aligned}$$

We can see that this works because if we plug them in then for $H'(a) < 0.5$, we know that if we add 0.5 to the loss function, then we achieve the same loss for $H(a) < 0.5$

and we know $H(a) = H'(a) + 0.5 for H'(a)) < 0.5$:

$$\begin{aligned} H'(a) + \lambda' \|w\|_2^2 &= H'(a) + (\lambda + \frac{0.5}{\|w\|_2^2})(\|w\|_2^2) \\ &= H'(a) + \lambda(\|w\|_2^2) + \frac{0.5}{\|w\|_2^2}(\|w\|_2^2) \\ &= H'(a) + \lambda(\|w\|_2^2) + 0.5 \\ &= H(a) + \lambda(\|w\|_2^2) \end{aligned}$$

Things are different for when $H'(a) > 0.5$ because we lose information about the loss value because we choose 0 for all values afterword. To compensate, we just add the loss function from before $H(a)$:

$$\begin{aligned} H'(a) + \lambda' \|w\|_2^2 &= H'(a) + (\lambda + \frac{max(1-a,0)}{\|w\|_2^2})(\|w\|_2^2) \\ &= 0 + \lambda(\|w\|_2^2) + \frac{max(1-a,0)}{\|w\|_2^2})(\|w\|_2^2) \\ &= \lambda(\|w\|_2^2) + max(1-a,0) \\ &= H(a) + \lambda(\|w\|_2^2) \end{aligned}$$

3) Kernel Trick (10 points) The kernel trick extends SVMs to handle with nonlinear data sets. However, an improper use of a kernel function can cause serious over-fitting. Consider the following kernels.

(a) Polynomial kernel: $K(x, x') = (1 + (xx'^T))^d$, where $d \in \mathbb{N}$. Does increasing d make over-fitting more or less likely?

Answer: As we increase d, we can imagine adding more degrees of freedom to the polynomial kernel. Thus, we are adding more curves for the boundary to take. This allows the boundary to curve more to the training samples to achieve a good margin. This kind of boundary overfits to the sample data because it takes a polynomial shape very specific to the training data. The more we increase d, the more we increase the flexibility of the boundary and thus over-fitting is more likely.

(b) Gaussian kernel: $K(x, x') = \exp(-||x - x'||^2/2\sigma^2)$, where $\sigma > 0$. Does increasing σ make over-fitting more or less likely?

Answer: The more we increase σ, the less over-fitting is likely. This is because we can imagine σ as the standard deviation of the separating boundary from the nearest support vector. The larger σ is, the larger this margin becomes and thus we have more confidience in our separating. As a result, the more we decrease the margin, the closer we allow support vectors to be toward the separating margin. This allows the boundary to be closer to the training sample points and thus not robust for test samples.

We say K is a kernel function, if there exists some transformation $\phi : \mathbb{R}^m \to \mathbb{R}^{m'}$ such that $K(x_i, x_{i'}) = \langle \phi(x_i), \phi(x_{i'}) \rangle$.

(c) Let K_1 and K_2 be two kernel functions. Prove that $K(x_i, x_{i'}) = K_1(x_i, x_{i'}) + K_2(x_i, x_{i'})$ is also a kernel function.

$$\begin{aligned}
K_1(x_i, x_{i'}) + K_2(x_i, x_{i'}) &= \langle \Phi_1(x), \Phi_1(x') \rangle + \langle \Phi_2(x), \Phi_2(x') \rangle \\
&= \Phi_1(x)^T \Phi_1(x') + \Phi_2^T(x), \Phi_2(x') \\
&= [\Phi_1(x)\Phi_2(x)][\Phi_1(x')\Phi_2(x')]^T \\
&= \Phi(x)^T \Phi(x') \\
&= \langle \Phi(x), \Phi(x') \rangle \\
&= K(x_i, x_{i'})
\end{aligned}$$

for $\Phi(x) = [\Phi_1(x)\Phi_2(x)]^T$

We know this from the definition of the kernel, which is given above. Then we can form a Φ that is a matrix concatenation of the two separate Φ_1 and Φ_2. This is adding new dimension to the sum kernel.

4) Prediction using Kernel (8 points) One of the differences between linear SVMs and kernel SVMs concerns computational complexity at prediction time.

(a) What is the computational complexity of prediction of a linear SVM in terms of the examples n and features m?

Answer: $O(m)$ because the decision rule only relies on the number of features because we are only doing a dot product to check the decision. When we predict using a linear SVM, we find the dot product of our model parameters w^T and the given m features x. This takes up the bulk of the computatio. $[w^Tx + b]$

(b) What is the computational complexity of prediction of a non-linear SVM in terms of the examples n, features m and support vectors s?

Answer: $O(sm)$ because we are inner producting ($O(m)$ time) and we are doing this s times for all the support vectors because when we are using a non-linear SVM, it is not always possible to store the hyperplane but more to store the support vectors. Thus we find the sum through all the support vectors of the dot product of the x_i for the support vector and x, the features of test point. $[\sum_i^N \alpha_i y_i(x_i^T x) + b\ ]$

5) Stochastic Gradient Algorithm (12 points) The stochastic gradient algorithm is a very powerful optimization tool to solve large-scale machine learning problems. Instead of computing the gradient over the entire data set before making an update, the stochastic gradient algorithm computes the gradient over a single sample, then updates the parameters. By passing over the entire data set of n samples in this fashion we can converge to the optimal parameters.

A single iteration of stochastic gradient considers a single example. While it takes many more iterations, each iteration is much faster, both in terms of memory and computation.

Consider a ridge regression problem with n samples:

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \|\beta\|_2^2. \tag{3}$$

In each iteration, instead of using only one example, we randomly choose k out of n samples and obtain $(x_{1'}, y_{1'}), ..., (x_{k'}, y_{k'})$.

(a) What is the computational complexity of computing the mini-batch stochastic gradient or gradient at each iteration (using k and n samples respectively)?

Answer: $O(km)$ where m is the number of features. This is because in mini-batch stochastic gradient we are working with k examples in each iteration. For each of these examples, we must find the dot product $(x_i^T \beta^2)$. Each example has $O(m)$ features.

Answer: $O(nm)$ because in this batch stochastic gradient, we consider all n instead of k (where $k \leq n$). We do the dot product for all n samples which have $O(m)$ features.

Note: for the entire algoirhtm, we reach a much worse running time for batch because before we take any steps, we must calculate the gradient for all n samples whereas for sgd, we only calculate the gradient for one point and then update the parameters.

(b) What are the advantages/disadvantages of increasing k in terms of computational complexity (using k and n samples respectively)? What is traded-off by increasing/decreasing k?

Answer:

When we increase k, we increase the number of comptutations in each iterations. This means that before we can update the parameters w, we must calculate the gradient for more k points rather than $k-1$ or 1 points. This leads to longer times to convergence because iterations are slower and so are steps. However, the more we increase k, the more accurate and better our steps get. This is true because we take the gradient of more sample points and thus we get a better idea of where the global (or better local) extrema are because we are getting an average gradient rather than step in the direction of a single sample's gradient.

This does not take into account vectorizing our sum. If we allow for parallelizing this summation then we can achieve a faster **running** time (not necessarily complexity time) because we can have the different iterations of the summation running at the same time.

Answer: The trade-off between increasing/decreasing k is that when we increase k, we have a larger computational complexity per iteration but better accuracy and less iterations. This is because we must check more gradients as we increase k per iteration before we update, but as a result we can get a better, more accurate idea of where the global (or better local) extrema are.

(c) Give one advantage and one disadvantage of using stochastic gradient descent with $k = 1$, ignoring computational considerations. Explain.

Answer: One disadvantage is that when you use stochastic gradient descent with $k = 1$, you are more susceptible to local minimums. This means that local extrema are more likely to make updates toward these local extrema and possibly away from better local minimum or even a global minimum. This is why it might never converge. However, if there is a lot of noise, then with a sgd with $k = 1$ the noise may help the parameters "jump" out of a local extrema and toward a better extrema.