CS 189: Introduction to Machine Learning - Discussion 5

1. General Review

a) Given a dataset, X, in $\mathbb{R}^n$, and the feature mapping ϕ corresponding to a quadratic kernel, what is the dimensionality of $\phi(X)$? What if the feature mapping was Gaussian/RBF? How do we deal with this high-dimensional data?

Solution: The quadratic kernel is defined as: $k(\mathbf{x},\ \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^2$.
$K(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^n x_i y_i + c\right)^2 =$
$\sum_{i=1}^n \left(x_i^2\right)\left(y_i^2\right) + \sum_{i=2}^n \sum_{j=1}^{i-1} \left(\sqrt{2}x_i x_j\right)\left(\sqrt{2}y_i y_j\right) + \sum_{i=1}^n \left(\sqrt{2c}x_i\right)\left(\sqrt{2c}y_i\right) + c^2$

$\Phi(\mathbf{z}) = \langle x_n^2, \ldots, x_1^2, \sqrt{2}x_n x_{n-1}, \ldots,$
$\sqrt{2}x_n x_1, \sqrt{2}x_{n-1}x_{n-2}, \ldots, \sqrt{2}x_{n-1}x_1, \ldots, \sqrt{2}x_2 x_1, \sqrt{2c}x_n, \ldots, \sqrt{2c}x_1, c\rangle$

So, the dimension is $1 + n + n + \binom{n}{2} = (N+2)(N+1)$, or approximately $\frac{N^2}{2}$.

The Gaussian kernel corresponds to an infinite dimensional feature mapping. Deriving the kernel expansion is a bit difficult, but if interested, it can be found on page 39 of `http://arxiv.org/pdf/0904.3664v1.pdf`. Although explicitly mapping our data into these high-dimensional spaces would be computationally infeasibly, kernels allow us to implicitly perform this feature map while still remaining in the original space.

b) Which types of learning problems have we seen so far? What types of models?

Solution: So far, all problems have been instances of supervised learning. We have covered both classification and regression problems.
Most of our models have been linear: Hebb's rule (classification), logistic regression (classification), linear regression (regression), and SVMs (classification), but we have learned how to deal with non-linear data through kernel methods.

c) What is Hebb's rule? How do we modify Hebb's rule to obtain logistic regression?

Solution: Hebb's rule is $\mathbf{w} = \sum_k \mathbf{y}^k \mathbf{x}^k$. We can represent logistic regression in a similar way. Define $\mathbf{z} = \mathbf{y} f(\mathbf{x}) = \mathbf{y} * \mathbf{wx}$. The logistic loss function, as given in lecture, is $L = \log(1 + e^{-\mathbf{z}})$. We would like to compute the derivative of this loss function with respect to the weight vector. Proceed by using the chain rule: $\frac{dL}{dw} = \frac{dL}{dz}\frac{dz}{dw}$.
$\frac{dL}{dz} = \frac{-e^{-\mathbf{z}}}{1+e^{-\mathbf{z}}}$.
$\frac{dz}{dw} = \mathbf{yx}$. Just like Hebb's rule, we update by this gradient. The inner term in the chain rule, $\frac{dz}{dw}$, is the exact same as in Hebb's rule. Here, however, the term is weighted by the sigmoid function!

2. Feature Selection

   Feature selection is the process of selecting a subset of a dataset's original features for usage in learning tasks. Any algorithm for this task has two parts: a search technique for proposing new feature subsets, as well as an evaluation measure which scores the different feature subsets. Wrapper methods involve retraining a model with these different subsets. Filter methods don't interact with the model, but rather just the features–modelling interactions between the features, and eliminating features that are highly correlated. Embedded methods embed the process of feature selection directly into the loss function.

   a) Why might we want to perform feature selection?

   Solution: There are a few reasons. By reducing the number of features, we can make models simpler and more interpretible. Imagine we had a rich dataset on people, and were trying to predict obesity. The most predictive features are height and weight (BMI), so ideally we would reduce the model to a function of these two variables, rather than a function of all of the collected features. This helps both to prevent overfitting, and allows the researcher to more easily interpret the result. We can also reduce training time.

   b) Describe the simplest wrapper method you can think of for performing feature selection. What is its runtime? Can we do better?

   Solution: Cross-validate models trained on every possible combination of features. This exhaustively checks all possible combinations. The runtime, unfortunately, is $O(2^d * t)$, where d is the number of features, and t is the time to train a model.

   c) Now, let's try the embedded method approach to feature selection.
      i) What kind of penalty term corresponds directly to feature selection?
      ii) What issues arise when optimizing a function with this penalty term?
      iii) Describe two vectors such that their ℓ-1 norms are equal, but their ℓ-2 norms are not.
      iv) Assume a method exists for optimizing non-differentiable functions, as long as they are convex.[1] How can we approximate the penalty term from part i, while still maintaining an "optimizable" loss function?

[1]These are known as proximal gradient methods:
`http://www.eecs.berkeley.edu/~elghaoui/Teaching/EE227A/lecture18.pdf`

Solution:

i. We would like to penalize the number of features considered. This is equivalent to penalizing the number of nonzero features in the weight vector: the $\ell - 0$ norm.

ii. The $\ell - 0$ norm is neither convex nor differentiable.

iii. $V_0 = (-0.5, -0.5), V_1 = (1, 0)$.
$\|V_0\|_1 = |-0.5| + |-0.5| = 1$
$\|V_1\|_1 | = |1| + |0| = 1.$
$\|V_0\|_2 = \sqrt{|-0.5|^2 + |-0.5|^2} = 0.71$
$\|V_1\|_2 = \sqrt{|1|^2 + |0|^2} = 1$

iv. Making use of the existing optimization method, we can use the ℓ-1 norm instead. This norm is piecewise linear, and convex. As you can see from the previous part, the ℓ-1 norm allows sparser solutions than the ℓ-2 norm, which penalizes high values in any one dimension.

Further Improvements Weicheng actually worked on these types of problems at Google this summer, and made use of two other types of norms, the truncated ℓ-1 norm, and the Lorentzian norm. These are descrsibed in detail at the following link: `http://people.csail.mit.edu/torralba/courses/6.869/lectures/lecture19/lecture19.pdf`

3. Virtual Leave One Out Cross-Validation[2]

K-Fold Cross Validation works pretty well, but imagine a case where our training data is really scarce. We could get even more usage out of our data by training on all data points except for one, then attempt to classify the remaining point. This corresponds to setting $K = n$ in K-Fold Cross Validation, and is called Leave-One-Out-Cross-Validation (LOOCV). Although this would seem to require n model fits, we'll derive a method which saves us this expense in the linear regression setting.

Note/Correction: This question is a bit simpler to derive without the regularization term, but is really motivated by cross-validating λ in Lasso/Ridge regression. Substituting the weight vector from ridge regression, $\hat{w}^T = (X^TX + \lambda * I)^{-1}X^Ty$, allows us to make use of the same LOOCV derivation in the ridge regression setting.

Let's write the formal definition of LOOCV:
$LOOCV = \sum_{i=1}^{n}(y_i - \hat{y_i}^{-i})^2$, where $\hat{y_i}^{-i}$ is the estimator of y_i with the i-th data point held-out when fitting the model.

a) After fitting our regression model, we have $\hat{y} = X\hat{w}^T$. With a tiny bit of algebra, we can rewrite our estimate of $\hat{y}$ in terms of the actual labels, $\hat{y} = Hy$, where H is an n-by-n matrix [3]. Write the equation for $\hat{y_i}$ in terms of H and y. Feel free to use the solution for $\hat{w}$ that we have found previously.

Solution: $\hat{y_i} = \sum_j H_{ij}y_j$.
Note: The optimal solution to linear regression, $\hat{w}^T$, is $(X^TX)^{-1}X^Ty$.
So, $\hat{y} = X\hat{w}^T = X(X^TX)^{-1}X^Ty = Hy$, and $H = X(X^TX)^{-1}X^T$.

b) By definition, $\hat{y}^{-i}$ minimizes $\sum_{j\neq i}(y_j - \hat{y_j}^{-i})^2$. Prove $\hat{y}^{-i}$ minimizes the squared error for Z where

$$Z_j = \begin{cases} y_j, & j \neq i \\ \hat{y_i}^{-i}, & i = j \end{cases}$$

Solution: From the definition, $\hat{y}^{-i} = argmin\sum_{j\neq i}(y_j - \hat{y_j}^{-i})^2$.
However, $z_i = \hat{y}_i^{-i}$.
So, $argmin\sum_{j\neq i}(y_j - \hat{y_j}^{-i})^2 = argmin\sum_j(z_j - \hat{y}_j^{-i})^2 = \hat{y}_i^{-i}$.

[2] Problem Set-Up: Credit to Tom Mitchell and Andrew W. Moore, CMU
[3] This matrix is known as the hat matrix, and as a number of useful statistical properties: `https://en.wikipedia.org/wiki/Hat_matrix`

c) Prove $\hat{y_i}^{-i} = \hat{y}_i - H_{ii}y_i + H_{ii}\hat{y_i}^{-i}$

Solution: From part a and b, $\hat{y}_i = \sum_j H_{ij}y_j$ and $\hat{y}_i = \sum_j H_{ij}z_j$.
Also, $\hat{y}_i - \hat{y}_i^{-i} = \sum_j H_{ij}(y_j - z_j) = H_{ii}(y_i - \hat{y}_i^{-i})$.
So, $\hat{y_i}^{-i} = \hat{y}_i - H_{ii}y_i + H_{ii}\hat{y_i}^{-i}$.

d) Using this result, show that $LOOCV = \sum_{i=1}^{n} (\frac{y_i - \hat{y}_i}{1 - H_{ii}})^2$. With this result, we can implement LOOCV while only fitting one regression model.

Solution: $LOOCV = \sum_{i=1}^{n} (y_i - \hat{y_i}^{-i})^2$.
Let's manipulate the result from part c: $\hat{y_i}^{-i}(1 - H_{ii}) = \hat{y}_i - H_{ii}y_i$.
Thus, $\hat{y_i}^{-i} = \frac{\hat{y}_i - H_{ii}y_i}{(1 - H_{ii})}$.
$LOOCV = \sum_{i=1}^{n} (y_i - \frac{\hat{y}_i - H_{ii}y_i}{(1 - H_{ii})})^2 = \sum_{i=1}^{n} (\frac{y_i - \hat{y}_i}{1 - H_{ii}})^2$.

e) Give an example of a dataset where LOOCV will give a poor estimate of the error.

Solution: In a dataset where every point is duplicated, the LOOCV will give an estimate of zero error. Since we only hold out one point at each step, there is a high likelihood that duplicates (or, at least, very near duplicates) may be contained in the training set. This results in inaccurately low error rates. As the dataset grows larger, the probability of near duplicates increases, and so LOOCV is best used in the small data setting.