CSE446 Machine Learning, Spring 2013: Homework 3

Due: Friday, May 17^{th}, beginning of class

Instructions There are 3 written questions on this assignment, plus a fourth coding question. Submit both your written answers (as a .txt or a .pdf) and your implementation to the Dropbox at `https://catalyst.uw.edu/collectit/assignment/darylh/26829/108643`

1 Error bound for 1-nearest neighbor classifier [30 points]

A nice result by Cover and Hart (1967) shows that, as the amount of training data approaches infinity, the error rate of the 1-nearest neighbor classifier is at most twice the Bayes-optimal error rate. In this problem, you will prove this result for the case of binary classification with real-valued inputs.

Let x^1, x^2, ... be the training examples and y^j be the corresponding binary class labels, $y^j \in \{0, 1\}$. You can think of x as points in some fixed d-dimensional Euclidean space.

Let $p_y(x) = p(X = x|Y = y)$ be the true conditional probability distribution for points in class y. We assume continuous and non-zero conditional probabilities: $0 < p_y(x) < 1$ for all x and y. Let also $\theta = p(Y = 1)$ be the probability that a random training example is in class 1. Again, assume $0 < \theta < 1$.

1. *(5 Points)* Given these expressions calculate the true probability $q(x) = p(Y = 1|X = x)$ that a data point x belongs to class 1. Express $q(x)$ in terms of $p_0(x)$, $p_1(x)$, and θ. (Hint: Use Bayes rule.)

2. *(5 Points)* A Bayes-optimal classifier is a classifier that always assigns a data point x the most probable class $\arg\max_y P(Y = y|X = x)$. This means Bayes-optimal classifier is maximizing the probability of correct classification. Given some test data point x, what is the probability that example x will be misclassified using the Bayes-optimal classifier, in terms of $q(x)$? (Hint: Start by splitting the cases where the true class 1, versus the true class is 0.)

3. *(5 Points)* Now the 1-nearest neighbor classifier assigns a test data point x the label of the closest training point x'. Given some test data point x and its nearest neighbor x', what is the expected error of the 1-nearest neighbor classifier, i.e., the probability that x will be misclassified, in terms of $q(x)$ and $q(x')$?

4. *(5 Points)* In the asymptotic case, the number of training examples of each class goes to infinity, and the training data fills the space in a dense fashion. Then the nearest neighbor x' of x has $q(x')$ converging to $q(x)$. By performing this substitution in the previous expression, give the asymptotic error for the 1-nearest neighbor classifier at point x, in terms of $q(x)$.

5. *(5 Points)* Show that the asymptotic error obtained in part 4 is less than twice the Bayes-optimal error obtained in part 2.

6. *(5 Points)* Why doesn't this asymptotic error bound hold in the non-asymptotic case, where the number of training examples is finite? You may want to draw a picture to illustrate your point here.

2 Learning Theory [5 Points]

If the Hypothesis space H is finite, you have m i.i.d. samples, and $0 < \epsilon < 1$ then for any learned hypothesis h we have:

$$P(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon) \leq |H|e^{-2N\epsilon^2}$$

If we fix some desired $\epsilon, \delta > 0$ such that:

$$P(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon) \leq |H|e^{-2N\epsilon^2} \leq \delta$$

Then with probability at least $1 - \delta$ we have:

$$\text{error}_{true}(h) - \text{error}_{train}(h) \leq \epsilon \tag{1}$$

But from the first bound:

$$|H|e^{-2N\epsilon^2} \leq \delta$$

1. *(2 Points)* Calculate the minimum required number of samples N in terms of ϵ and δ such that (1) holds.

2. *(3 Points)* Consider using decision trees with fixed depth k then the number of decision trees is bounded by:

   $$2 \cdot (2n)^{2^k - 1}$$

   Calculate the minimum required number of samples N if you wish to use decision trees of depth $k = 4$ and $\delta = 0.05$ and $\epsilon = 0.05$ you have $n = 10$ features.

3 Fitting an SVM classifier by hand [30 Points]

(Source: Murphy text, Exercise 14.1) Consider a dataset with 2 points in 1d: $(x_1 = 0, y_1 = -1)$ and $(x_2 = \sqrt{2}, y_2 = 1)$. Consider mapping each point to 3d using the feature vector $\phi(x) = [1, \sqrt{2}x, x^2]^T$. (This is equivalent to using a second order polynomial kernel.) The max margin classifier has the form

$$min||w||^2 \qquad s.t. \tag{2}$$

$$y_1(w^T\phi(x_1) + w_0) \geq 1 \tag{3}$$

$$y_2(w^T\phi(x_2) + w_0) \geq 1 \tag{4}$$

1. *(6 Points)* Write down a vector that is parallel to the optimal vector w. Hint: recall from Figure 14.12 (page 500 in the Murphy text) that w is perpendicular to the decision boundary between the two points in the 3d feature space.

2. *(6 Points)* What is the value of the margin that is achieved by this w? Hint: recall that the margin is the distance from each support vector to the decision boundary. Hint 2: think about the geometry of 2 points in space, with a line separating one from the other.

3. *(6 Points)* Solve for w, using the fact the margin is equal to $1/||w||$.

4. *(6 Points)* Solve for w_0 using your value for w and Equations 2 to 4. Hint: the points will be on the decision boundary, so the inequalities will be tight. A "tight inequality" is an inequality that is as strict as possible. For this problem, this means that plugging in these points will push the left-hand side of Equations 3 and 4 as close to 1 as possible.

5. *(6 Points)* Write down the form of the discriminant function $f(x) = w_0 + w^T \phi(x)$ as an explicit function of x. Plot the 2 points in the dataset, along with $f(x)$ in a 2d plot. You may generate this plot by hand, or using a computational tool like R or Matlab.

4 Programming Question [35 Points]

In this problem, we seek to perform a digit recognition task, where we are given an image of a handwritten digit and wish to predict what number it represents. This is a special case of an important area of language processing known as Optical Character Recognition (OCR). We will be simplifying our goal to that of a binary classification between two relatively hard-to-distinguish numbers (specifically, predicting a '3' versus a '5'). To do this, you will implement a kernelized version of the Perceptron algorithm.

4.1 Dataset

The digit images have been taken from the Kaggle competition linked to on the projects page, `http://www.kaggle.com/c/digit-recognizer/data`. This data was originally from the MNIST database of handwritten digits, but was converted into a easier-to-use file format.

The data has also undergone some preprocessing. It has been filtered to just those datapoints whose labels are 3 or 5, which have then been relabeled to 1 and -1 respectively. Then, 1000-point samples have been created, named *validation.csv* and *test.csv*. The first column of these files is the label of each point, followed by the grayscale value of each pixel.

4.2 Perceptron

In the basic Perceptron algorithm, we keep track of a weight vector w, and define our prediction to be $\hat{y}^{(t)} = sign(w \cdot x^{(t)})$. If we predict a point correctly, we make no update and continue running. Any time we make a mistake, our update step is

$$w^{(t+1)} \leftarrow w^{(t)} + y^{(t)} x^{(t)},$$

$$\text{so at time } t,\ w^{(t)} = \sum_{i \in M^{(t)}} y^i x^i$$

where $M^{(t)}$ is the set of mistakes made up to time t.

4.3 Kernels

To apply the kernel trick, we would like to replace x and w with $\Phi(x)$ and $\Phi(w)$, where $\Phi : X \to F$ is a mapping into some high- or infinite-dimensional space. For example, Φ could map to the set of all polynomials of degree exactly d. To do this, we try to find a function k for this particular Φ that has the property $k(u, v) = \Phi(u) \cdot \Phi(v)$ for every u and v. The trick, however, is that although this function lets us compute dot products easily, we must not actually deal with any $\Phi(x)$ directly. Because of this, the natural extension of storing our weight vector doesn't work:

$$\Phi(w^{(t)}) = \sum_{i \in M^{(t)}} y^i \Phi(x^i)$$

would require both computing the sum of $\Phi(x^i)$ explicitly and storing it as $\Phi(w)$. As stated above, $\Phi(w)$ could have millions (or in fact an infinite number) of terms, so this can quickly become impractical. Instead, we can rely on the fact that our prediction becomes

$$\hat{y}^{(t)} = sign(\Phi(w) \cdot \Phi(x)) = sign\left(\sum_{j \in M^{(t)}} y^j \Phi(x^j) \cdot \Phi(x^{(t)})\right) = sign\left(\sum_{j \in M^{(t)}} y^j k(x^j, x^{(t)})\right)$$

This means that it's possible for us to store the (x^j, y^j) pairs of our mistakes $M^{(t)}$, and use these to compute our dot product $\Phi(w) \cdot \Phi(x)$.

The drawback of this is that we are now storing a list of all mistakes we ever make, which is quite a bit more overhead than simply w in the case without kernels. This also means that if we make too many mistakes, performing the dot product can become quite slow. However, we are now able to build much more complex models, and changing between models is as easy as switching kernel functions.

4.4 Task

Hint: It is probably a good idea to write your Perceptron implementation so that it can be passed a kernel function as an argument. If you wish to apply a function to each row of a matrix in R, the built-in "apply" function is much more efficient than using a loop.

1. *(15 Points)* First, get Perceptron working with the kernel $k_p^1(u, v) = u \cdot v + 1$.
   (k_p^1 is what the standard dot product would give us, if we had added a constant term $x_0 \equiv 1$.)
   Run Perceptron for a single pass over the validation set with this kernel, and plot the average loss $\bar{L}$ as a function of the number of steps T, where

$$\bar{L}(T) = \frac{1}{T} \sum_{j=1}^{T} \mathbb{1}(\hat{y}^{(t)} \neq y^{(t)}) \tag{5}$$

   where $\hat{y}^t$ is the label that Perceptron predicts for datapoint t as it runs, and $\mathbb{1}$ is an indicator function, which is 1 if its condition is true and 0 otherwise. Record the average loss every 100 steps, e.g. [100, 200, 300, ...].

2. *(10 Points)* For a positive integer d, the polynomial kernel $k_p^d(u, v) = (u \cdot v + 1)^d$ maps X into a feature space of all polynomials of degree up to d.

   For the set $d = [1, 3, 5, 7, 10, 15, 20]$, run Perceptron for a single pass over the validation set with k_p^d, and plot the average loss over the validation set $\bar{L}(1000)$ as a function of d. What value of d produces the lowest loss?

3. *(10 Points)* For $\sigma > 0$, the Exponential kernel $k_E^\sigma(u, v) = e^{-\frac{\|u-v\|}{2\sigma^2}}$ is a map to *all* polynomials of x, where σ is a tuning constant that roughly corresponds to the "window size" of the exponential. Tuning on the validation set has produced a value of $\sigma = 10$.

   For the d you chose in the previous step, run Perceptron with both k_p^d and k_E^{10} for a single pass over the testing set. For each of these two kernels, plot the average loss $\bar{L}(T)$ as a function of the number of steps. As above, report the average loss for every 100 steps.