MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.867 MACHINE LEARNING, FALL 2006

Problem Set 2

Due Date: Thursday, Oct 12, 12:00 noon
You may submit your solutions in class or in the box.

1. The data and scripts for this problem are available in hw2/prob1. You can load the data using the MATLAB script `load_al_data`. This script should load the matrices `y_noisy, y_true, X_in`. The y vectors are $n \times 1$ while `X_in` is a $n \times 3$ matrix with each row corresponding to a point in $\mathcal{R}^3$.

   The y_{true} vectors correspond to the ideal y values, generated directly from the "true" model (whatever it may be) *without any noise*. In contrast, the y_{noisy} vectors are the actual, noisy observations, generated by adding Gaussian noise to the y_{true} vectors. You should use y_{noisy} for any estimation. y_{true} is provided only to make it easier to evaluate the error in your predictions (simulate an infinite test data). You would not have y_{true} in any real task.

   (a) Write MATLAB functions `theta = linear_regress(y,X)` and `y_hat = linear_pred(theta,X_test)`. Note that we are *not* explicitly including the offset parameter but instead rely on the feature vectors to provide a constant component. See part (b).

   (b) The feature mapping can substantially affect the regression results. We will consider two possible feature mappings:

$$\begin{aligned}\phi_1(x_1, x_2, x_3) &= [1, x_1, x_2, x_3]^T \\ \phi_2(x_1, x_2, x_3) &= [1, \log x_1^2, \log x_2^2, \log x_3^2]^T\end{aligned}$$

   Use the provided MATLAB function `feature_mapping` to transform the input data matrix into a matrix

$$X = \begin{bmatrix} \phi(\mathbf{x_1})^T \\ \phi(\mathbf{x_2})^T \\ \cdots \\ \phi(\mathbf{x_n})^T \end{bmatrix}$$

   For example, `X = feature_mapping(X_in,1)` would get you the first feature representation. Using your completed linear regression functions, compute the mean squared prediction error for each feature mapping (2 numbers).

   (c) The selection of points to query in an active learning framework might depend on the feature representation. We will use the same selection criterion as in the lectures, the expected squared error in the parameters, proportional to $Tr[(X^TX)^{-1}]$. Write a MATLAB function `idx = active_learn(X,k1,k2)`. Your function should assume that the top k_1 rows in X have been queried and your goal is to sequentially find the indices of the next k_2 points to query. The final set of $k_1 + k_2$ indices should be returned in `idx`. The latter may contain repeated entries. For each feature mapping, and $k_1 = 5$ and $k_2 = 10$, compute the set of points that should be queried (i.e., `X(:,idx)`). For each set of points, use the feature mapping ϕ_2 to perform regression and compute the resulting mean squared prediction errors (MSE) over the entire data set (again, using ϕ_2).

(d) Let us repeat the steps of part (c) with randomly selected additional points to query. We have provided a MATLAB function `idx = randomly_select(X,k1,k2)` which is essentially the same as `active_learn` except that it selects the k_2 points uniformly at random from X. Repeat the regression steps as in previous part, and compute the resulting mean squared prediction error again. To get a reasonable comparison you should repeat this process 50 times, and use the median MSE. Compare the resulting errors with the active learning strategies. What conclusions can you draw?

(e) Let us now compare the two sets of points chosen by active learning due to the different feature representations. We have provided a function `plot_points(X,idx_r,idx_b)` which will plot each row of `X` as a point in $\mathbf{R}^3$. The points indexed by `idx_r` will be circled in red and those marked by `idx_b` will be circled (larger) in blue (some of the points indexed by `idx_r` and `idx_b` might be common). Plot the original data points using the indexes of the actively selected points based on the two feature representations. Also plot the same indexes using `X` from the second feature representation with its first constant column removed. In class, we saw an example where the active learning strategy chose points at the extrema of the available space. Can you see evidence of this in the two plots?

2. We derived a kernel version of linear regression in class using regularized least squares criterion. The resulting predictions were given by

$$y(\mathbf{x}) = \sum_{t=1}^{n} (\hat{\alpha}_t/\lambda) K(\mathbf{x}_t, \mathbf{x})$$

where the optimal setting of the coefficients (prediction differences) $\hat{\alpha}_t$ were given in a vector form by

$$\mathbf{a} = \lambda(\lambda \mathbf{I} + \mathbf{K})^{-1}\mathbf{y}$$

Here $\mathbf{K}$ is the Gram matrix based on the available data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and $\mathbf{y} = [y_1, \ldots, y_n]^T$. We are interested in exploring how the regularization parameter $\lambda \in [0, \infty)$ affects the solution when the kernel function is the radial basis kernel

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\beta}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right), \quad \beta > 0$$

a) Let's see first that the limit $\lambda \to 0$ (no regularization) makes sense. In this case we need the Gram matrix $\mathbf{K}$ to be invertible. Indicate why the Gram matrix corresponding to the radial basis kernel is always invertible based on the following *Michelli theorem:*

If $\rho(t)$ is a monotonic function of $t \in [0, \infty)$ then the matrix $\rho_{ij} = \rho(\|\mathbf{x}_i - \mathbf{x}_j\|)$ is invertible for any distinct set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

b) How do we make predictions when $\lambda \to 0$? In other words, what is the function $y(\mathbf{x})$ in this limit?

b) Show that the corresponding training error is exactly zero.

c) Setting $\lambda = 0$ (no regularization) seems hardly optimal. Implement the kernel linear regression method above for $\lambda > 0$. We have provided training and test data as well as helpful MATLAB scripts in hw2/prob2. You should only need to complete the relevant lines in `run_prob2` script. The data pertains to the problem of predicting Boston housing prices based on various indicators (normalized). Evaluate and plot the training and test errors (mean squared errors) as a function of λ in the range $\lambda \in (0, 1)$. Use $\beta = 0.05$. Explain the qualitative behavior of the two curves.

3. Most linear classifiers can be turned into a kernel form. We will focus here on the simple perceptron algorithm and use the resulting kernel version to classify data that are not linearly separable.

(a) First we need to turn the perceptron algorithm into a form that involves only inner products between the feature vectors. We will focus on hyper-planes through origin in the feature space (any offset component provided as part of the feature vectors). The mistake driven parameter updates are: $\theta \leftarrow \theta + y_t \phi(\mathbf{x}_t)$ if $y_t \theta^T \phi(\mathbf{x}_t) < 0$, where $\theta = 0$ initially. Show that we can rewrite the perceptron updates in terms of simple additive updates on the *discriminant function* $f(\mathbf{x}) = \theta^T \phi(\mathbf{x})$:

$$f(\mathbf{x}) \leftarrow f(\mathbf{x}) + y_t K(\mathbf{x}_t, \mathbf{x}) \ \text{ if } y_t f(\mathbf{x}_t) < 0$$

where $K(\mathbf{x}_t, \mathbf{x}) = \phi(\mathbf{x}_t)^T \phi(\mathbf{x})$ is any kernel function and $f(\mathbf{x}) = 0$ initially.

(b) We can replace $K(\mathbf{x}_t, \mathbf{x})$ with any kernel function of our choice such as the radial basis kernel where the corresponding feature mapping is infinite dimensional. Using the analysis in the previous question (2), show that there always is a separating hyperplane if we use the radial basis kernel.

(c) With the radial basis kernel we can therefore conclude that the perceptron algorithm will converge (stop updating) after a finite number of steps for any dataset with distinct points. The resulting function can therefore be written as

$$f(\mathbf{x}) = \sum_{i=1}^{n} w_i y_i K(\mathbf{x}_i, \mathbf{x})$$

where w_i is the number of times we made a mistake on example $\mathbf{x}_i$. Most of w_i's are exactly zero so our function won't be difficult to handle. The same form holds for any kernel except that we can no longer tell whether the w_i's remain bounded (problem is separable with the chosen kernel).

Implement the new kernel perceptron algorithm in MATLAB using a radial basis and polynomial kernels. The data and helpful scripts are provided in hw2/prob3.

Define functions
`alpha = train_kernel_perceptron(X, y, kernel_type)` and
`f = discriminant_function(alpha, X, kernel_type, X_test)`
to train the pereptron and to evaluate the resulting $f(\mathbf{x})$ for test examples, respectively.

(d) Load the data using the `load_p3_a` script. When you use a polynomial kernel to separate the classes, what degree polynomials do you need? Draw the decision boundary (see the provided script `plot_dec_boundary`) for the lowest-degree polynomial kernel that separates the data. Repeat the process for the radial basis kernel. Briefly discuss your observations.