10-701/15-781, Machine Learning: Homework 3

Eric Xing, Tom Mitchell, Aarti Singh
Carnegie Mellon University
Updated on February 7, 2010

- The assignment is due at 10:30am (beginning of class) on **Mon, Feb 22, 2010**.
- Separate you answers into three parts, one for each TA, and put them into 3 piles at the table in front of the class. Don't forget to put both your name and a TA's name on each part.
- If you have question about any part, please direct your question to the respective TA who design the part.
- Submit your code to HW3 in the blackboard $http://www.cmu.edu/blackboard$

1 Linear regression, and bias-variance trade-off[20pt, Ni Lao]

In linear regression, we are given training data of the form, $\mathcal{D} = (\mathbf{X}, \mathrm{y}) = \{(\mathrm{x}_i, y_i)\}, i = 1, 2, ..., n$, where $\mathrm{x}_i \in \mathcal{R}^{1\times p}$, i.e. $\mathrm{x}_i = (x_{i,1}, \cdots, x_{i,p})^{\mathrm{T}}$, $y_i \in \mathcal{R}$, $\mathbf{X} \in \mathcal{R}^{n\times p}$, where n is number of samples, p is number of features. Each row i of $\mathbf{X}$ is ${\mathrm{x}_\mathrm{i}}^{\mathrm{T}}$, and $\mathrm{y} = (y_1, \cdots, y_n)^{\mathrm{T}}$.

Least square regression seeks to find β that minimizes the square-error, i.e.:

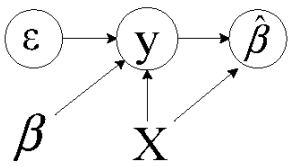
$$\mathrm{J}_1(\beta) = \sum_i (y_i - {\mathrm{x}_\mathrm{i}}^{\mathrm{T}}\beta)^2 = (\mathbf{X}\beta - \mathrm{y})^{\mathrm{T}}(\mathbf{X}\beta - \mathrm{y}).$$

It has an unique solution

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathrm{y}. \tag{1}$$

where $\hat{\beta}$ is called an estimator of β. An "estimator" is a statistic of your data (i.e. a function of your data) which is intended to approximate a parameter of the underlying distribution. There is a research field called "estimation theory", which deals with constructing estimators that have nice properties, like converging to the correct parameter given enough data, and giving confidence intervals. In this problem we will explore how regularization affects the bias and variance of the least square regression model.

Let's assume that $p < n$, and $\mathbf{X}^T\mathbf{X}$ is invertible. Also assume that our data is generated from a true model of the form: $y_i = {\mathrm{x}_\mathrm{i}}^{\mathrm{T}}\beta + \epsilon_i$ (or in matrix form $\mathrm{y} = \mathbf{X}\beta + \epsilon$), where $\epsilon_1, ..., \epsilon_n$ are IID and sampled from a Gaussian with 0 mean and constant standard deviation, that is $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ (or $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$). Relations among these quantities can be summarized by the figure below, where quantities with circles are random variables. Here we assume that $\mathbf{X}$ is "deterministic" and fixed.



1.1 Least square regression [4 pt]

Show that $\hat{\beta}$ has Gaussian distribution and write down its mean μ and covariance matrix Σ. You will see that least square regression is unbiased $E[\hat{\beta}] = \beta$

Hint: if $\epsilon \sim \mathcal{N}(0, I)$ then $A\epsilon \sim \mathcal{N}(0, AA^T)$, where A is any matrix.

Hint: notation is simpler, if you do a Singular Value Decomposition (SVD) to $\mathbf{X}$ first.

1.2 Ridge regression [4pt]

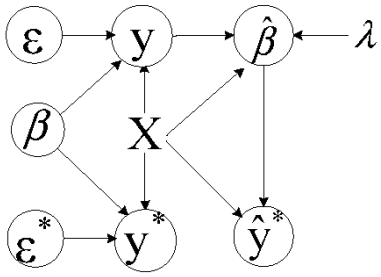
The solution to ridge regression is

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathrm{y}. \tag{2}$$

Show that $\hat{\beta}$ has Gaussian distribution and write down its mean μ and covariance matrix Σ. You will see that ridge regression is biased $E[\hat{\beta}] \neq \beta$.

1.3 The bias variance trade-off [4 pt]

Now let's be real Bayesians, and believe that the true parameter β itself is a random variable. Let's assume that $\beta \sim \mathcal{N}(0, \alpha^2 I)$. Apart from our training data $\mathcal{D} = (\mathbf{X}, \mathrm{y})$, we also generate a set of testing data $\mathcal{D}^* = (\mathbf{X}, \mathrm{y}^*)$. It has exactly the same x values $\mathbf{X}$ as training data, but the y values are regenerated independently. Again we can decompose them as $\mathrm{y}^* = \mathbf{X}\beta + \epsilon^*$. Relations among these quantities can be summarized by the figure below.



Now we want see how should we choose the regularization parameter λ so that the risk of ridge regression on test data $\mathcal{D}^*$ is minimized.

As a first step, let's express $\mathrm{e}(\lambda) = \hat{\mathrm{y}}^* - \mathrm{y}^*$, the test errors of ridge regression on $\mathcal{D}^*$, in terms of β, ϵ, and ϵ^*. You will see three terms: one grows as λ grows corresponding to the bias of our estimation, one diminishes as λ grows corresponding to the variance of our estimation, and one that is independent of λ corresponding to the irreducible error.

1.4 [4 pt]

Express the risk of ridge regression $R(\lambda) = E[\mathrm{e}(\lambda)^T\mathrm{e}(\lambda)] = \sum_i E[e_i^2(\lambda)]$ in terms of α, σ, and λ.

Hint: if $a \sim \mathcal{N}(0, ASA^T)$, where A is unitary, S is diagonal, then $E[a^Ta] = \sum_i S_{i,i} = tr(S)$.

Hint: if a and b are independent random vectors with zero means, then $E[(a+b)^T(a+b)] = E[a^Ta] + E[b^Tb]$

1.5 [4 pt]

Find the optimal regularization parameter λ, that minimizes the risk of ridge regression on test data $\mathcal{D}^*$. You will see that the larger the magnitude of data noise ϵ (controlled by σ) and the

smaller the magnitude of the true parameter β (controlled by α) the larger regularization is needed to achieve the minimum risk.

Hint: for simplicity, you can assume that the number of features $p = 1$. The result should still hold for $p > 1$.

Hint: the result is an expression of σ and α.

2 Neural Networks: Learning and Representation [30 pt, Amr]

As we discussed in class, one important question we need to ask when learning about a new classifier is what kind of decision boundaries can this classifier learn. We have explored this question in HW1 for the case of 1-KNN and decision trees, and showed that both of them can vary their decision boundary either on a data-driven way for 1-KNN or based on the size of the tree for decision trees. In this question we will explore these issues for the case of a 2-layer Neural Network (NN). Recall from class that the input a_j for a node j is given by:

$$a_j = \sum_i w_{ji} o_i$$

Where, w_{ji} is the weight from unit i to unit j, and o_i is the activatoin/output of unit i. The activation of unit i is the output of a logistic function in this problem (although any differentiable function is allowed):

$$o_i = \sigma(a_i) = \frac{1}{1 + \exp^{(-a_i)}}$$

2.1 Decision Boundary

Consider the classification task shown in figure 1 where '+' and 'o' denotes positive and negative classes, respectively. This data is available in the file `data.mat, also please read NN_readme.txt which contains a simple code to draw the figure below`. For this part of the problem, you might want to write really a few lines of matlab code to evaluate the network and get the necessary plots. Consider the 2-layer network in Figure 1. This network has 9 weights and a logistic activation function for both the hidden and output layers.

(1) **[2 points]** For each of the following classifiers, state with a one-line explanation whether or not they can learn the decision boundary illustrated in Figure 1: 1-KNN, decision tree, Naive Bayes, and logistic regression.

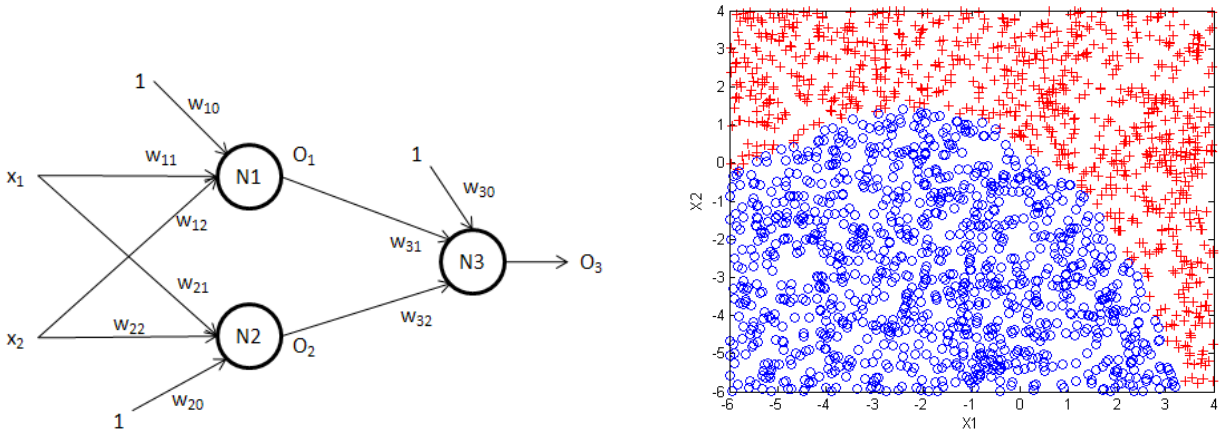
(2) **[1 point]** Express o_1 and o_2 in terms of $x_1, x_2, w_{10}, w_{11}, w_{12}, w_{20}, w_{21}, w_{22}$.

(3) **[1 point]** Write down the decision rule for this 2-layer NN classifier.

(4) Consider the following set of weights: $w_{10} = -0.8; w_{11} = 0.8; w_{12} = 0.1; w_{20} = 0.3; w_{21} = 0.3; w_{22} = -0.4; w_{31} = 1.0; w_{32} = -1.0; w_{30} = 0.2;$

   (a) **[2 points]** Draw the representation of the data after the hidden layer, i.e. your dimensions will be o_1 and o_2, and you should label each point with its ground-truth label. State your observation.

   (b) **[2 points]** Draw the classification result for this 2-layer NN, i.e. re-draw the scatter plot in Figure 1 but label each point as classified by the 2-layer NN. Compare to the ground-truth and state your observations.



(a) (b)

Figure 1: (a) 2-layer NN with logistic activation functions at both the hidden and output layers. (b) A 2-class dataset: '+' and 'o' marks positive and negative labels respectively.

(c) **[2 points]** Overlay the decision boundary **explicitly** over the curve your drew in part (a) in terms of o_1 and o_2. State your observations.

(e) Lets assume that we removed the logistic function form the hidden layer ONLY and instead used an identity function, i.e $o_1 = a_1$ and $o_2 = a_2$, while maintaining the same weight values.

(i) **[2 points]** repeat (a) and (b) and state your observations.

(ii) **[2 points]** Can you tweak the weights in this case to learn the correct decision boundary? If yes, then find such weights and repeat (i), if NO, then re-express the network in this case using a simpler network (or classifier) and argue why it can not learn this decision boundary.

2.2 Backpropagation

In class we discussed that in order to train a NN we need to define an error function E[W] (such as the squared error) that can be minimized using the backpropagation algorithm to find the network weights. We also discussed that this error function can be driven using the M(C)LE principle based on a signal plus noise interpretation in the context of a regression setting. Consider a classification task with Data $\mathcal{D} = (\mathbf{X}, \mathbf{t}) = \{(\mathrm{x}^i, t^i)\}, i = 1, 2, ..., N,$. For example, $\mathbf{x}^i$ might be a face image, and t^i is a binary label equals 0 if the face is for a male and 1 if the face is for a female. Now consider a 2-layer NN based on logistic threshold units at both the hidden and output layers. If we let y denote the real-valued final output of the network, where $y \in [0, 1]$, then we might naturally wish to interpret this output as the probability that the boolean class label t takes on the value $t = 1$; that is, $y = P(t = 1|x; W)$. In this case, as we have done in logistic regression, it is natural to find the NN weights W using the M(C)LE principle as follows:

$$\mathbf{W}_{\mathrm{MLE}} = \arg\max_{\mathbf{w}} \prod_{i=1}^{N} p(t^i|\mathrm{x}^i; \mathbf{w}) \tag{3}$$

(1) **[2 points]** Show that maximizing (3) is equivalent to minimizing the cross-entropy error function given by:

$$\mathrm{E}[W] = -\sum_{i=1}^{N} \left[t^i \ln y^i + (1 - t^i) \ln(1 - y^i) \right]$$

where, y^i is the output of the network corresponding to example i.

Now we will drive the weight update rule using a stochastic gradient decent using a random example n at each step, thus we will add n to the error function, E_n to make this dependency explicit. The generic update rule for a weight w_{ji} coming from unit (or input i) to unit j is given by:

$$w_{ji}^{\text{new}} = w_{ji}^{\text{old}} - \eta \frac{\partial \mathrm{E}_n}{\partial w_{ji}^{\text{old}}}$$

where η is the learning rate. Lets denote the output unit as k, therefore, $o_k = y$. For this part of the problem, assume there are no bias terms.

(2) **[2 points]** Define $\delta_j = \frac{\partial \mathrm{E}_n}{\partial a_j}$. Show that $\frac{\partial \mathrm{E}_n}{\partial w_{ji}} = \delta_j o_i$

(3) **[4 points]** Show that for the output unit $\delta_k = y - t$, where t is the correct label, and $o_k = y \in [0, 1]$ is the network prediction.

(4) **[2 points]** Write down the update rule for a weight between the output unit k and a hidden unit j.

(5) **[4 points]** For a hidden unit j, use the chain rule to show that, $\delta_j = \frac{\partial \mathrm{E}_n}{\partial a_j} = \sigma(a_j)(1 - \sigma(a_j))(y - t)w_{kj}$. [**Hint**: $\frac{\partial \mathrm{E}_n}{\partial a_j} = \frac{\partial \mathrm{E}_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$, where k is the output unit.]

(6) **[2 points]** Write down the update rule for a weight between the hidden unit j and input x_i.

3 Clustering [50 pt, Field Cady]

Clustering means partitioning your data into "natural" groups, usually because you suspect points in a cluster have something in common. The EM algorithm and k-means are two common algorithms (there are many others). This problem will have you implement these algorithms, and explore their limitations.

The datasets for you to use are available online, along with a Matlab script for loading them. Ask me if you're having any trouble with it. Instructions for submitting code will be posted later.

You can use any language for your implementations, but you may not use libraries which already implement these algorithms (you can, however, use fancy built-in mathematical functions, like Matlab or Mathematica provide).

3.1 K-means : implementation [15 pt]

In k-means clustering, the goal is to pick your clusters such that you minimize the sum, over all points x, of $|x - c_x|^2$, where c_x is the mean of the cluster containing x; this should remind you of

least-squares line fitting. K-means clustering is NP-hard, but in practice Lloyd's algorithm, also called the "k-means algorithm", works extremely well.

Implement Lloyd's algorithm, and apply it to the datasets provided. Plot each dataset, indicating for each point which cluster is was placed in.

3.2 K-means : evaluation [5 pt]

How well do you think k-means did for each dataset? Explain, intuitively, what (if anything) went badly and why.

3.3 EM Algorithm : implementation [25 pt]

A disadvantage of k-means is that the clusters cannot overlap at all. Expectation maximization deals with this by only probabilistically assigning points to clusters.

The thing to understand about the EM algorithm is that it's a special case of MLE; you have some data, you assume a parameterized form for the probability distribution (a mixture of Gaussians is, after all, an exotic parameterized probability distribution), and then you pick the parameters to maximize the probability of your data. But the usual MLE approach, solving $\frac{\partial P(X|\theta)}{\partial \theta} = 0$, isn't tractable, so we use the iterative EM algorithm to find θ. The EM algorithm is guaranteed to converge to a local optimum (I'm resisting the temptation to make you prove this :)).

Implement the EM algorithm, and apply it to the datasets provided. Assume that the data is a mixture of two Gaussians; you can assume equal mixing ratios. What parameters do you get for each dataset? Plot each dataset, indicating for each point which cluster it was placed in.

3.4 EM Algorithm : evaluation [5 pt]

Modeling dataset 2 as a mixture of gaussians is unrealistic, but the EM algorithm still gives *an* answer. Is there anything "fishy" about your answers which suggests something is wrong?

We usually do the EM algorithm with mixed Gaussians, but you can use any distributions; a Gaussian and a Laplacian, three exponentials, etc. Write down the formula for a parameterized probability density suitable for modeling "ring-shaped" clusters in 2D; don't let the density be 0 anywhere. You don't need to work out the EM calculations for this density, but you would if this came up in your research.

3.5 Extra Credit [5 pt]

With high-dimensional data we cannot perform visual checks, and problems can go unnoticed if we assume nice round, filled clusters. Describe in words a clustering algorithm which works even for weirdly-shaped clusters with unknown mixing ration. However, you can assume that the clsuters do not overlap at all, and that you have a LOT of training data. Discuss the weaknesses of your algorithm. Don't work out the details for this problem; just convince me that you know the basic idea and understand its limitations. Please keep your answers concise.

Hint : Nothing we've covered so far in class will help you here; think about graphs...