10-701 Machine Learning, Fall 2011: Homework 2

Due 10/17 at the beginning of class.

1 Linear regression, model selction [25pt, Nan Li]

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 ${\mathrm{x_i}}^{\mathrm{T}}$, and $\mathrm{y} = (y_1, \cdots, y_n)^{\mathrm{T}}$.

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

$$\mathrm{J}_1(\theta) = \sum_i (y_i - {\mathrm{x_i}}^{\mathrm{T}}\theta)^2 = (\mathbf{X}\theta - \mathrm{y})^{\mathrm{T}}(\mathbf{X}\theta - \mathrm{y}).$$

It has an unique solution

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

where $\hat{\theta}$ is called an estimator of θ. 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: $y_i = \mathrm{x}_i\theta + \epsilon_i$ (or in matrix form $\mathrm{y} = \mathbf{X}\theta + \epsilon$), 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 $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$).

Let's first check how regularization could change the bias and variance of the estimator $\hat{\theta}$.

1.1 Ridge regression [10 pt]

We know that the original linear regression without any regularization is unbiased, i.e. $E[\hat{\theta}] = \theta$. What about ridge regression? The solution to ridge regression is

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

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

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 Extra features [10 pt]

Now, let's move on to see the relation between model complexity and overfitting. Can we keep increasing our model quality by adding more features in it?

The objective of least square regression is to minimize $\mathrm{J}_1(\theta)$ given $\mathcal{D}$. Assume that besides the existing feature data, $\mathbf{X}$, we have, we now have access to one more new feature. The feature values for the given samples are X_{new}, where $X_{new} \in \mathcal{R}^{n\times 1}$. Taken together we have a new feature matrix $\mathbf{X}_{new} = [\mathbf{X}\ X_{new}]$. For simplicity, let's assume $\mathbf{X}_{new}^T\mathbf{X}_{new} = I$. Show that with this new feature, the minimized objective value $\mathrm{J}_1(\theta)$ will be decreased.

1.3 Model Complexity [3 pt]

Does the above result mean that as long as we keep increasing features, we can always get a better model? Briefly explain why or why not.

1.4 Overfitting [2 pt]

Please suggest one method to address possible overfitting reflected in the above example.

2 Learning Theory [25 points + 3 extra points, Bin]

2.1 PAC Learning

PAC stands for "Probably Approximately Correct" and concerns a nice formalism for deciding how much data you need to collect in order for a given classifier to achieve a given probability of correct predictions on a given fraction of future test data.

- **[5 points]** True or false: (if true, give a 1 sentence justification; if false, give a counter example.) Within the setting of PAC learning, it is **impossible** to assure with probability 1 that the concept will be learned perfectly (i.e., with true error = 0), regardless of how many training examples are provided.

2.2 VC Dimension

For hypothesis class $\mathcal{H}$, its VC dimension is at least d if there exists some samples $|S| = d$ which is shattered by $\mathcal{H}$. Please note that this does not mean all samples of size d need to be shattered by $\mathcal{H}$. To show that the VC dimension is at most d, one must show that no sample of size $d+1$ could be shattered by $\mathcal{H}$. In this problem, you will calculate the VC-dimension of some hypothesis classes. Remember you need to show the following 2 steps to prove the VC dimension of $\mathcal{H}$ is d:

- There exists a set of d points which can be shattered by $\mathcal{H}$;
- There exists no set of $d+1$ points which can be shattered by $\mathcal{H}$.

You need to show details of how you get the VC dimension in order to get full credit.

(1) **[4 points]** $\mathcal{X} = \mathbb{R}$, $\mathcal{H}$ is the union of 2 intervals;

(2) **[4 points]** $\mathcal{X} = \mathbb{R}^2$, $\mathcal{H}$ is the set of axis-parallel squares (with equal height and width);

(3) **[4 points]** $\mathcal{X} = \mathbb{R}$, $\mathcal{H}$ is the set of functions defined as following:

$$h(x) = \text{sign}(\sin(ax + b)) \tag{3}$$

where a and b are parameters in h, $\text{sign}(x) = +1$ if $x \geq 0$ and $\text{sign}(x) = -1$ otherwise.

(4) **[3 extra points]** $\mathcal{H}$ is a finite hypothesis class such that $|\mathcal{H}| < \infty$. Show that the VC dimension of $\mathcal{H}$ is upper bounded by $\log_2 |\mathcal{H}|$. (Hint: how many different labelings are there for a set of size $\log_2 |\mathcal{H}| + 1$?)

2.3 The Approximation-Estimation-Optimization Tradeoff

In this problem, we consider a space of input-output pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ generated by a probability distribution $P(x, y)$. We define a loss function $l(\hat{y}, y)$ (for example, $l(\hat{y}, y) = |\hat{y} - y|^2$ as in regression), to measure the discrepancy between the predicted output $\hat{y}$ and the real output y. Our target is to find the function f^* that minimizes the expected risk

$$R(f) = \int l(f(x), y) dP(x, y) \tag{4}$$

Usually the distribution $P(x, y)$ is unknown, we are instead given a sample $\mathcal{S}$ of n independently drawn training examples $\{(x_i, y_i)\}$, $i = 1, \ldots, n$. We define the empirical risk

$$R_n(f) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i) \tag{5}$$

The learning principle we learned in class consists of first choosing a family $\mathcal{F}$ of candidate prediction functions, then finding the function $f_n = \arg\min_{f \in \mathcal{F}} R_n(f)$. Since the optimal function f^* is usually unlikely to belong to the family $\mathcal{F}$, we also define $f^*_{\mathcal{F}} = \arg\min_{f \in \mathcal{F}} R(f)$. For simplicity, we assume that f^*, $f^*_{\mathcal{F}}$ and f_n are well defined and unique. We can then decompose the excess error as

$$\mathbb{E}[R(f_n) - R(f^*)] = \mathbb{E}[R(f^*_{\mathcal{F}}) - R(f^*)] + \mathbb{E}[R(f_n) - R(f^*_{\mathcal{F}})] = \varepsilon_{app} + \varepsilon_{est} \tag{6}$$

where the expectation is taken with respect to the random choice of training set. The approximation error ε_{app} measures how closely functions in $\mathcal{F}$ can approximate the optimal solution f^*. The estimation error ε_{est} measures the effect of minimizing the empirical risk $R_n(f)$ instead of the expected risk $R(f)$.

One flaw of the above decomposition of excess error is that it assumes we find f_n that minimizes the empirical risk $R_n(f)$. However, this procedure is often a computationally expensive operation. Let us assume that our minimization algorithm returns an approximate solution $\tilde{f}_n$ that minimizes the objective function up to a predefined tolerance $\rho \geq 0$

$$R_n(\tilde{f}_n) < R_n(f_n) + \rho \tag{7}$$

We can then decompose the excess error $\varepsilon = \mathbb{E}[R(\tilde{f}_n) - R(f^*)]$ as

$$\varepsilon = \mathbb{E}[R(f^*_{\mathcal{F}}) - R(f^*)] + \mathbb{E}[R(f_n) - R(f^*_{\mathcal{F}})] + \mathbb{E}[R(\tilde{f}_n) - R(f_n)] = \varepsilon_{app} + \varepsilon_{est} + \varepsilon_{opt} \tag{8}$$

We call the additional ε_{opt} the optimization error. It reflects the impact of the approximate optimization on the generalization performance.

(1) **[8 points]** In this question, you will study how approximation error ε_{app}, estimation error ε_{est}, optimization error ε_{opt} and computation time T change when one of $\{\mathcal{F}, n, \rho\}$ increases. (In creasing $\mathcal{F}$ means that the new $\mathcal{F}$ is a superset of the old $\mathcal{F}$) Fill in table 1 with $\uparrow$ indicating increase, $\downarrow$ indicating decrease, and $\times$ indicating not affected. Please briefly explain your answer.

Table 1: Typical variations when $\mathcal{F}$, n and ρ increase.

	$\mathcal{F}$	n	ρ
ε_{app}			$\times$
ε_{est}			$\times$
ε_{opt}	$\times$	$\times$	
T			

3 Expectation-Maximization algorithm [Suyash, 25 points]

In this question, we will derive some details about the expectation-maximization (EM) algorithm and work out an example application.

3.1 True-False Questions [6 points]

Explain whether the following statements are true or false with a single line of explanation.

1. **(2 points)** The EM algorithm maximizes the complete log-likelihood.
2. **(2 points)** Even if the complete log-likelihood is multi-modal, EM cannot get stuck in a local minima.
3. **(2 points)** The free-energy functional that EM optimizes is a lower bound on the complete log-likelihood.

3.2 EM and KL divergence [7 points]

3.2.1 Optimizing the free-energy functional [4 points]

Consider the free-energy functional discussed in class that the EM algorithm optimizes. The free-energy functional is given by:

$$F(q, \theta) = \sum_{q(z|x)} q(z|x) \log \frac{p(z, x|\theta)}{q(z|x)} \tag{9}$$

for any probability distribution $q(z)$. (Note: In this setting, the only random variables are the z variables). In class, we claimed that

$$\log p(x|\theta) - F(q, \theta) = KL(q \,\|\, p(z|x; \theta)) \tag{10}$$

where $KL(a \,\|\, b) = \sum_x a(x) \log \frac{a(x)}{b(x)}$ is called the Kullback-Leibler divergence between two probability distributions a and b. Show that this result is true.

3.2.2 Optimal value of q(z) in EM [3 points]

Using the result in Equation 10, find the value for $q(z)$ that maximizes $F(q, \theta)$. (Hint: $KL(a \,\|\, b) \geq 0$, with equality if and only if $a = b$.)

3.3 Example of coin-tossing [12 points]

Consider the scenario where a sequence of heads and tails are being generated by tossing two coins independently. For each toss, the first coin is chosen with probability π and the second coin is chosen with probability $1-\pi$. The head probabilities of the two coins are given by p_1 and p_2. The probabilities π, p_1 and p_2 are unknown to us and we want to find their values.

Let us represent the result of the N coin tosses by $x_1, x_2, \cdots, x_N$. $x_i = 1$ indicates that that i^{th} toss came out heads and 0 otherwise. Let $z_1, z_2, \cdots, z_N$ indicate the coin which was tossed each time. For ease of mathematical analysis, suppose that $z_i = 1$ means the first coin was used for the i^{th} toss and $z_i = 0$ means the second coin was used for the i^{th} toss.

3.3.1 All variables observed

Suppose that we are allowed to observe which coin was used for each toss, i.e, the values of the variables $z_1, \cdots, z_N$ are known. The expression for the complete log-likelihood of the N coin tosses is

$$\begin{aligned} l_c(X, Z; \pi, p_1, p_2) &= \log \prod_{i=1}^{N} p(x_i, z_i; \pi, p_1, p_2) \\ &= \sum_{i=1}^{N} \log \left[\pi p_1^{x_i}(1-p_1)^{1-x_i}\right]^{z_i} \left[(1-\pi) p_2^{x_i}(1-p_2)^{1-x_i}\right]^{1-z_i} \\ &= \sum_{i=1}^{N} [z_i(\log \pi + x_i \log p_1 + (1-x_i)\log(1-p_1)) \\ &\qquad + (1-z_i)(\log(1-\pi) + x_i \log p_2 + (1-x_i)\log(1-p_2))] \end{aligned}$$

and the maximum likelihood equations for the parameters in this setting are:

$$\pi = \frac{\sum_{i=1}^{N} z_i}{N} \tag{11}$$

$$p_1 = \frac{\sum_{i=1}^{N} z_i x_i}{\sum_{i=1}^{N} z_i} \tag{12}$$

$$p_2 = \frac{\sum_{i=1}^{N} (1-z_i) x_i}{\sum_{i=1}^{N} (1-z_i)}. \tag{13}$$

3.3.2 Only x variables observed

Suppose we are not allowed to observe which coin was used for each toss, i.e, the value of the variables $z_1, \cdots, z_N$ are unknown. We will now use the EM algorithm for finding the values of the unknown parameters.

1. **(3 points)** Write the expression for the incomplete log-likelihood.

2. **(5 points)** The E-step for our EM algorithm requires us to compute $q^*(z) = p(z|x; \pi^{(t)}, p_1^{(t)}, p_2^{(t)})$ where the superscript t indicates the value of the parameters at step t. What is the expression for the conditional probability $p(z_i = 1|x_i; \pi^{(t)}, p_1^{(t)}, p_2^{(t)})$ for the random variable z_i associated with the i^{th} toss? Note that your expression should only involve $\pi^{(t)}, p_1^{(t)}, p_2^{(t)}$ and x_i. For notational convenience, you can drop the superscript t from your expression.

3. **(4 point)** We can show that the updates for the parameters π, p_1 and p_2 in the M-step are given by

$$\pi^{(t+1)} = \frac{\sum_{i=1}^{N} E[z_i|x;\pi^{(t)},p_1^{(t)},p_2^{(t)}]}{N} \tag{14}$$

$$p_1^{(t+1)} = \frac{\sum_{i=1}^{N} E[z_i|x;\pi^{(t)},p_1^{(t)},p_2^{(t)}]x_i}{\sum_{i=1}^{N} E[z_i|x;\pi^{(t)},p_1^{(t)},p_2^{(t)}]} \tag{15}$$

$$p_2^{(t+1)} = \frac{\sum_{i=1}^{N} (1-E[z_i|x;\pi^{(t)},p_1^{(t)},p_2^{(t)}])x_i}{\sum_{i=1}^{N} (1-E[z_i|x;\pi^{(t)},p_1^{(t)},p_2^{(t)}])} \tag{16}$$

How are these updates different from the maximum-likelihood expressions derived for the parameters in the setting where all the variables are observed?

4 K-means (programming) [Qirong Ho, 25 points + 5 bonus points]

The k-means algorithm is one of the most popular unsupervised machine learning tools, thanks to a combination of simplicity, speed, effectiveness, and perhaps most importantly, visual intuitiveness. Many of us can "see" how k-means works, but what is it really doing mathematically? It turns out that k-means is connected to the following problem: given an integer k and a set of d-dimensional data points $x_1, \ldots, x_n$, pick k d-dimensional vectors $c_1, \ldots, c_k$ (referred to as "centers") that minimize the potential function

$$\psi = \sum_{i=1}^{n} \min_{j \in \{1,\ldots,k\}} \|x_i - c_j\|^2 .$$

The potential function ψ measures the Euclidean distance between every point x_i and its closest center c_j. Minimizing ψ essentially involves picking centers c such that every data point x is close to some center — a fairly intuitive notion of clustering. Unfortunately, exactly minimizing ψ happens to be NP-hard (don't worry if you don't know what that means). We will soon prove that k-means finds a local minimum of ψ — in other words, k-means doesn't necessarily find the best solution to ψ, but we hope it finds a good one at least.

First, recall the k-means algorithm:

1. Choose initial values for the centers $c_1, \ldots, c_k$.
2. For each data point x_i, let z_i be the index of its closest center, e.g. $z_i = j$ means c_j is the closest center to x_i.
3. For each center c_j, set c_j to be the mean of the data points closer to it than any other center. In other words, set $c_j = \frac{\sum_{i=1}^{n} \delta(z_i=j)x_i}{\sum_{i=1}^{n} \delta(z_i=j)}$, where $\delta(z_i = j) = 1$ if $z_i = j$, and 0 otherwise.
4. Repeat steps 2 and 3 until ψ no longer changes.

4.1 K-means finds a local minimum of ψ

Let's convince ourselves that the k-means algorithm finds a local minimum of ψ. We shall focus on a single iteration of k-means. Consider the alternative potential function

$$\phi = \sum_{i=1}^{n} \sum_{j=1}^{k} \delta(z_i = j) \|x_i - c_j\|^2 .$$

Notice that ϕ is almost equivalent to ψ, except that the $\min_{j \in \{1,\ldots,k\}}$ in ψ has been replaced by the closest center assignments z_i. The key insight is that k-means performs a coordinate-wise minimization of ϕ: in step 2, we minimize ϕ with respect to the closest center assignments z_i (while holding c_j fixed), and then in step 3, we minimize ϕ with respect to the centers c_j (while holding z_i fixed).

1. [**4 points**] Assume the c_j are fixed to their values from the previous iteration. Show that step 2 minimizes ϕ with respect to all closest center assignments z_i. **Hint:** if you don't know where to begin, we're really asking you to prove the following statement: if you change one or more of the $z_1, \ldots, z_n$ produced by step 2, then the value of ϕ will either increase or stay the same. Notice that the double summation in ϕ contains nk terms; you should begin by separating them into k groups, one for each z_i.

2. [**6 points**] Now assume that the z_i are fixed to their values from step 2. Show that step 3 minimizes ϕ with respect to all centers c_j. **Hint:** try differentiating ϕ with respect to a single c_j.

The alternating minimization on z_i and c_j guarantees that k-means finds a local minimum of ϕ. Moreover, the values of c_j at this local minimum of ϕ (which is a function of z and c) also correspond to a local minimum of ψ (which is a function of c only). We thus conclude that k-means finds a local minimum of ψ.

4.2 Implementing k-means

Having convinced ourselves that k-means does something sensible, we shall now evaluate its performance on the dataset `faces.csv`. Each of the 2429 lines in this file corresponds to a 19×19 image of a human face. Every image is represented as a 361-dimensional vector of grayscale values, in column-major format (i.e. the first 19 numbers correspond to the first column, then the next 19 numbers correspond to the second column, etc.). If you are using MATLAB, you can load the images using '`data = csvread('faces.csv')`', and from there you can display the i-th image using '`imshow(uint8(reshape(data(i,:),[19 19])))`'.

1. [**5 points**] Implement the k-means algorithm as described earlier. Note that there are two additional issues we need to consider: how to choose the initial centers $c_1, \ldots, c_k$, and how to decide if convergence has been reached in step 4. For this part, we shall pick the initial centers *uniformly at random* (with replacement) from the set of data points $x_1, \ldots, x_n$. As for convergence, we are not going to use a stopping criterion — rather, we shall visualize the performance of k-means by plotting the alternative potential ϕ at every iteration (thus, your code will need to compute ϕ).

2. [**5 points**] Run your implementation 10 times on the `faces.csv` dataset, using $k = 5$ centers and taking 50 iterations for each run (one iteration is a single repeat of steps 2-3). Save the centers learned by each run, you'll need them for the next question! Plot ϕ versus iteration number for all 10 runs on the same graph. What do you observe?

3. [**5 points**] From your results in the previous part, find the run that gives the lowest value of ϕ at the last (50th) iteration. This is your algorithm's best run. Using your favorite plotting software, draw the $k = 5$ centers from this run as 19×19 grayscale images. On MATLAB, the command '`imshow(uint8(reshape(c,[19 19])))`' will plot the 361-dimensional vector c as a 19×19 grayscale image. What do you observe? Do the 'center-faces' look like faces at all?

Submit all written parts and plots as hardcopies at the beginning of lecture, and submit all code by email to the grading TA (qho AT andrew DOT cmu DOT edu). In particular, you are to provide scripts that generate all plots.

4.3 Bonus Question: choosing good initial centers with the k-means++ algorithm

In the paper "k-means++: The Advantages of Careful Seeding" (Arthur and Vassilvitskii 2007), the authors describe a probabilistic initialization scheme for the centers c_j. This scheme guarantees that, in expectation, the potential function ϕ will take a good value right from the start. The k-means++ initialization proceeds as follows:

1. Choose an initial center c_1 uniformly at random from the data points $x_1, \ldots, x_n$.
2. Choose the next center c_j probabilistically, such that the probability of choosing $c_j = x_i$ is $\frac{D(x_i)^2}{\sum_{h=1}^{n} D(x_h)^2}$, where $D(x)$ is the Euclidean distance from x to its nearest center in $\{c_1, \ldots, c_{j-1}\}$.
3. Repeat step 2 until all k centers have been chosen.

We want to compare this initialization scheme with the uniform random initialization from earlier.

1. [**Bonus 5 points**] Implement the k-means++ initialization, and repeat parts 2 and 3 from question 4.2. Compare your new plots to your old ones. What do you observe?

Again, please email scripts for generating your plots.