

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.867 MACHINE LEARNING, FALL 2009

Problem Set 3

Due Date: Monday, October 26 (at 5pm)

NOTE: you will not receive a graded problem set in return for submitting one. We have adopted a new grading scheme for problem sets. We will help you understand and work through the problems. You are therefore strongly encouraged to clear up any misunderstandings prior to the deadline. By submitting your solutions, you indicate a) that you do understand the problem and b) that you have worked through the solution. Your submission will be recorded and you will receive no other grade from the submission. By completing a problem set, you demonstrate an achievement. In total, problem sets count 30% of the course grade.

Electronic submission is required! A submission form will be made available at the course website. You should have received the password required for submission via email.

Please see the course website for help on finding and using Matlab. Note that you will need the Optimization Toolbox, which is included in the MIT distribution.

Problem 1 Collaborative filtering

We can think of collaborative filtering as matrix factorization. Any $n \times m$ matrix X can be approximated by a lower rank matrix UV^T where U is $n \times d$, V is $m \times d$, and $d \leq \min\{m, n\}$. Let $\underline{u}_i \in \mathcal{R}^d$ be the i^{th} row vector of U and \underline{v}_j the j^{th} row vector of V . Then $[UV^T]_{ij} = \underline{u}_i \cdot \underline{v}_j$. We want to use this low rank matrix to reproduce (approximate) observed binary labels in Y . Let $Y_{ij} \in \{-1, 1\}$ if the (i, j) entry is observed, and $Y_{ij} = 0$ otherwise. Then we would like find U and V such that

$$Y_{ij}[UV^T]_{ij} = Y_{ij}(\underline{u}_i \cdot \underline{v}_j) > 0 \quad \text{whenever } Y_{ij} \neq 0 \quad (1)$$

Note that this is trivially possible to achieve if $X = UV^T$ has full rank (each element can be chosen independently of each other). The smaller d we choose, the fewer adjustable parameters we have in U and V . d therefore seems to control the complexity of a solution that satisfies the constraints. We would find the smallest d for which the constraints are possible to satisfy.

It is often advantageous to potentially avoid satisfying all the constraints, e.g., if some of the labels may be mistakes. We might also suspect that we gain something by turning the estimating problem into a maximum margin problem, for \underline{u}_i 's given $\{\underline{v}_j\}$ and for \underline{v}_j 's given $\{\underline{u}_i\}$. The formulation with slack is given by

$$\begin{aligned} \min \quad & \sum_{i=1}^n \frac{1}{2} \|\underline{u}_i\|^2 + \sum_{j=1}^m \frac{1}{2} \|\underline{v}_j\|^2 + C \sum_{i,j: Y_{ij} \neq 0} \xi_{ij} \\ \text{subject to} \quad & Y_{ij}(\underline{u}_i \cdot \underline{v}_j) \geq 1 - \xi_{ij}, \quad \xi_{ij} \geq 0, \quad \text{for all } (i, j) \text{ for which } Y_{ij} \neq 0 \end{aligned} \quad (2)$$

Note that, if we fix U , i.e., fix all $\underline{u}_1, \dots, \underline{u}_n$, the problem reduces to m independent maximum margin SVM problems, one for each \underline{v}_j with labeled examples $\{(\underline{u}_i, Y_{ij})\}_{i: Y_{ij} \neq 0}$. The problem for each \underline{u}_i given V is defined analogously.

1.1 Suppose we have no observations in the i^{th} row. In other words, $Y_{ij} = 0$ for all $j = 1, \dots, m$. What is the resulting \underline{u}_i ? Will adding such an empty row affect what we get for $\{\underline{v}_j\}$?

1.2 Suppose there's only one observation in the i^{th} row, say $Y_{ij} = 1$ (the rest are zeros). Assume no slack ($C = \infty$). What is the resulting \underline{u}_i ? Note that the resulting \underline{u}_i specifies a maximum margin boundary through origin.

1.3 Suppose there are exactly two observations for each row. Provide an expression for \underline{u}_i on the basis of two labeled observations, say Y_{i1} and Y_{i2} associated with \underline{v}_1 and \underline{v}_2 . Again, assume no slack ($C = \infty$).

1.4 Consider a simple two-user and two-movie example with an observed rating matrix

$$Y = \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} \quad (3)$$

To predict the missing entry we can start with an initial guess for V and then solve for U and V iteratively, fixing one, then solving for the other (starting with U). The MATLAB routine `cf(Y,V,C,verbose)` solves eq. (2) iteratively while also illustrating the intermediate steps. Try

```
[U,V] = cf([1 -1;-1 0],[1 0;0 1],100,1); % slack penalty C=100, verbose = 1
[U,V] = cf([1 -1;-1 0],[1 1;0 1],100,1);
```

Does the solution you find depend on the initial conditions? Do the predicted labels flip in the course of the iterative steps? Try giving a rank deficient V as an initial guess. What do you find?

1.5 Consider the problem of filling in ratings of 100 users on 40 movies. Our data is extracted from the MovieLens¹ dataset. Each user rating is converted into a binary label (+1 or -1) depending on whether the rating exceeds a given threshold. The resulting binary labels are stored in two matrices Y and T of size 100×40 . Both matrices contain missing entries (indicated by zeros). We use Y for training and T for testing the CF algorithm. None of the observed entries of Y appear in T . The performance is evaluated based on how well we recover the labels in T .

The matrices Y and T are stored in the file `movie.mat`. You can use the above MATLAB routine in this context as well though it is not an efficient method for solving large factorization problems. You can use it, for example, as follows

```
C = 0.1; d = 2;
[U,V] = cf(Y,randn(size(Y,2),d),C);
X = U*V';
I = find(Y); disp(mean(Y(I).*X(I)<=0)); % training error
I = find(T); disp(mean(T(I).*X(I)<=0)); % test error
```

Try different values of slack penalty $C = 0.1, 0.5, 1$ for a fixed $d = 2$. Consider then $d = 2, 3, 4$ for a fixed $C = 0.1$. Which parameter, C or d , is more effective in terms of complexity control? Explain your reasoning.

Problem 2 Boosting

Boosting is a simple procedure for estimating ensembles

$$h_m(\underline{x}) = \sum_{j=1}^m \alpha_j h(\underline{x}; \theta_j) \quad (4)$$

¹<http://www.grouplens.org/node/73>

where $\alpha_j \geq 0$ (not necessarily summing to one). The base learners $h(\underline{x}; \theta_j)$ are often simple (weak) classifiers such as decision stumps. There are many variations on the boosting algorithm due to the choice of base learners or the loss function that the overall algorithm seeks to minimize.

At stage m of the algorithm, we we will try to minimize

$$J(\alpha_m, \theta_m) = \sum_{t=1}^n \text{Loss}\left(y_t h_m(\underline{x}_t)\right) = \sum_{t=1}^n \text{Loss}\left(y_t h_{m-1}(\underline{x}_t) + y_t \alpha_m h(\underline{x}_t; \theta_m)\right) \quad (5)$$

where we assume that $h_{m-1}(\underline{x})$ is fixed from $m - 1$ previous rounds and view J as a function of parameters α_m and θ_m associated with the new base classifier. The margin loss $\text{Loss}(\cdot)$ could be any function that is convex and decreasing in its argument (smaller loss for predictions with larger voting margin).

Let's elaborate on the algorithm a bit. We can initialize $h_0(\underline{x}) = 0$ for the ensemble with no base learners. Then, after $m - 1$ rounds, we perform two distinct steps to optimize θ_m and α_m , respectively. First, we find parameters $\hat{\theta}_m$ that minimize

$$\left. \frac{\partial}{\partial \alpha_m} J(\alpha_m, \theta_m) \right|_{\alpha_m=0} = \sum_{t=1}^n \overbrace{\text{dL}\left(y_t h_{m-1}(\underline{x}_t)\right)}^{-W_{m-1}(t)} y_t h(\underline{x}_t; \theta_m) = - \sum_{t=1}^n W_{m-1}(t) y_t h(\underline{x}_t; \theta_m) \quad (6)$$

where $\text{dL}(z) = d/dz \text{Loss}(z)$ is always negative because the loss is decreasing. (We applied the chain rule of derivatives to get the above form.) The values of the weights will be differ based on different loss functions but they will always decrease as a function of how well the current ensemble classifies the training examples. We can also normalize the weights in the algorithm since the normalization does not affect the resulting $\hat{\theta}_m$.

Once we have $\hat{\theta}_m$, we can find $\hat{\alpha}_m$ that minimizes

$$J(\alpha_m, \hat{\theta}_m) = \sum_{t=1}^n \text{Loss}\left(y_t h_{m-1}(\underline{x}_t) + y_t \alpha_m h(\underline{x}_t; \hat{\theta}_m)\right) \quad (7)$$

We usually will not get a closed-form expression for $\hat{\alpha}_m$. However, the optimization problem is easy to solve since $\text{Loss}(\cdot)$ is convex and we only have one parameter to optimize.

We can now write the general boosting algorithm more succinctly. After initializing $\tilde{W}_0(t) = 1/n$, each boosting iteration consists of the following three steps:

(Step 1) Find $\hat{\theta}_m$ that (approximately) minimizes the weighted error ϵ_m or $2\epsilon_m - 1$ given by

$$- \sum_{t=1}^n \tilde{W}_{m-1}(t) y_t h(\underline{x}_t; \theta_m) \quad (8)$$

(Step 2) Find $\hat{\alpha}_m$ that minimizes

$$J(\alpha_m, \hat{\theta}_m) = \sum_{t=1}^n \text{Loss}\left(y_t h_{m-1}(\underline{x}_t) + y_t \alpha_m h(\underline{x}_t; \hat{\theta}_m)\right) \quad (9)$$

(Step 3) Reweight the examples

$$\tilde{W}_m(t) = -c_m \text{dL}\left(y_t h_{m-1}(\underline{x}_t) + y_t \hat{\alpha}_m h(\underline{x}_t; \hat{\theta}_m)\right) \quad (10)$$

where c_m normalizes the new weights so that they sum to one.

Now that we have a boosting algorithm for any loss function, we can select a particular one. Specifically, we will consider the logistic loss:

$$\text{Loss}(z) = \log(1 + \exp(-z)) \quad (11)$$

2.1 Show that the unnormalized weights $W_m(t)$ from the logistic loss are bounded by 1. What can you say about the resulting normalized weights for examples that are clearly misclassified in comparison to those that are just slightly misclassified by the current ensemble?

2.2 Suppose the training set is linearly separable and we would use a hard-margin linear support vector machine (no slack) as a base classifier. In the first boosting iteration, what would the resulting $\hat{\alpha}_1$ be?

2.3 We have provided you with boosting code that you can run to test how it works. `mod = boost(X,y,ncomp)` generates an ensemble (a cell array of `ncomp` base learners) based on training examples `X` and labels `y`. `load data.mat` gives you `X` and `y` for a simple classification task. You can then generate the ensemble with any number of components (e.g., 50). The cell array `mod` simply lists the base learners in the order in which they were found. You can therefore plot the ensemble corresponding to the first `i` base learners by `plot_decision(mod(1:i),X,y)`, or individual base learners via `plot_decision({mod{i}},X,y)`.

How complex a classifier can we get by combining stumps into an ensemble? Show that any n distinct 1-dimensional points can be correctly classified by $2n$ stumps. (*Hint: use 2 stumps to construct a little bump around each point*).

2.4 `plot_voting_margin(mod,X,y,th)` helps you study how the voting margins change as a function of boosting iterations. For example, the plot with `th = 0` gives the fraction of correctly classified training points (voting margin > 0) as a function of boosting iterations. You can also plot the curves for multiple thresholds at once as in `plot_voting_margin(mod,X,y,[0,0.05,0.1,0.5])`. Explain why some of these tend to increase while others decrease as a function of boosting iterations. Why does the curve corresponding to `th = 0.05` continue to increase even after all the points are correctly classified?