CSE 546 Machine Learning, Autumn 2013
Homework 2

Due: Monday, October 28, beginning of class

1 Boosting [25 Points]

We learned about boosting in lecture and the topic is covered in Murphy 16.4. On page 555 Murphy claims that "it was proved that one could boost the performance (on the training set) of any weak learner arbitrarily high, provided the weak learned could always perform slightly better than chance." We will now verify this in the AdaBoost framework.

1. (*7 points*) Given a set of N observations (x^j, y^j) where y^j is the label $y^j \in \{-1, 1\}$, let $h_t(x)$ be the weak classifier at step t and let α_t be its weight. First we note that the final classifier after T steps is defined as:

$$H(x) = sgn\left\{\sum_{t=1}^{T} \alpha_t h_t(x)\right\} = sgn\{f(x)\}$$

Where

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Show that:

$$\epsilon_{\text{Training}} = \frac{1}{N}\sum_{j=1}^{N} 1_{\{H(x^j) \neq y^j\}} \leq \frac{1}{N}\sum_{j=1}^{N} \exp(-f(x^j)y^j)$$

Where $1_{\{H(x^j) \neq y^j\}}$ is 1 if $H(x^j) \neq y^j$ and 0 otherwise.

2. (*7 points*) The weight for each data point j at step $t+1$ can be defined recursively by:

$$w_j^{(t+1)} = \frac{w_i^{(t)} \exp(-\alpha_t y^j h_t(x^j))}{Z_t}$$

Where Z_t is a normalizing constant ensuring the weights sum to 1:

$$Z_t = \sum_{j=1}^{N} w_j^t \exp(-\alpha_t y^j h_t(x^j))$$

Show that:

$$\frac{1}{N}\sum_{j=1}^{N}\exp(-f(x^j)y^j) = \prod_{t=1}^{T} Z_t$$

3. (*11 points*) We showed above that training error is bounded above by $\prod_{t=1}^{T} Z_t$. At step t the values $Z_1, Z_2, \ldots, Z_{t-1}$ are already fixed therefore at step t we can choose α_t to minimize Z_t. Let

$$\epsilon_t = \sum_{j=1}^{m} w_j^t 1_{\{h_t(x^j)\neq y^j\}}$$

be the weighted training error for weak classifier $h_t(x)$ then we can re-write the formula for Z_t as:

$$Z_t = (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)$$

   (a) First find the value of α_t that minimizes Z_t then show that

$$Z_t^{opt} = 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

   (b) Assume we choose Z_t this way. Then re-write $\epsilon_t = \frac{1}{2} - \gamma_t$ where $\gamma_t > 0$ implies better than random and $\gamma_t < 0$ implies worse than random. Then show that:

$$Z_t \leq \exp(-2\gamma_t^2)$$

   You may want to use the fact that $\log(1-x) \leq -x$ for $0 \leq x < 1$

   Thus we have:

$$\epsilon_{\text{training}} \leq \prod_{t=1}^{T} Z_t \leq \exp(-2\sum_{t=1}^{T}\gamma_t^2)$$

   (c) Finally, show that if each classifier is better than random (e.g. $\gamma_t \geq \gamma$ for all t and $\gamma > 0$) that:

$$\epsilon_{\text{training}} \leq \exp(-2T\gamma^2)$$

Which shows that the training error can be made arbitrarily small with enough steps.

2 KNN [15 Points]

Consider the following 2D dataset: positive examples at (2,2), (2,4), (2,6), (2,8), and (4,5); negative examples at (4,3), (4,7), (6,4), (6,6), and (8,5).

1. *(3 points)* Draw a Voronoi diagram of this dataset using Euclidian distance.
2. *(2 points)* List the examples that could be removed without changing the frontiers produced by 1NN.
3. *(5 points)* Compute the error rate of 3NN using LOOCV. That is, for each point, remove it, and predict its class using the other examples. If some example has more than 3 examples at the same nearest distance such that different choices of the nearest neighbors give different predictions, then count the example as an error. This computes the worst-case error rate.

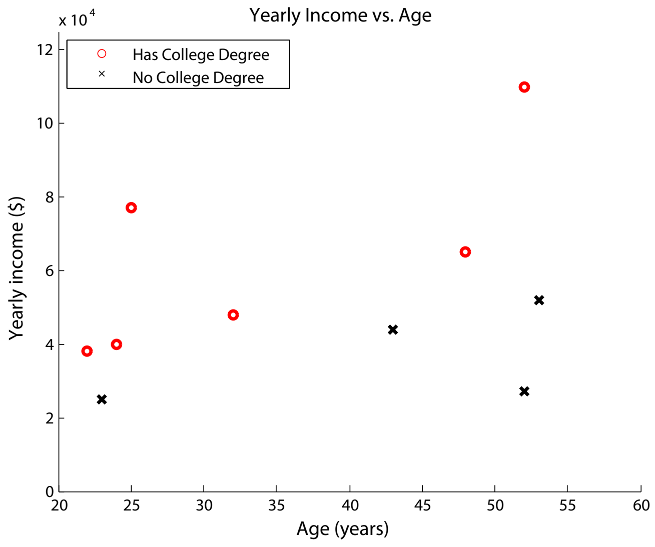
4. *(5 points)* One greedy algorithm for feature selection is called *backward elimination*. A learning algorithm is applied using a full set of D features, and its error rate is recorded. One by one for each feature, the algorithm is applied with (only) that feature excluded. This gives D new runs of the algorithm, each with an error rate. The feature whose exclusion produced the lowest error rate is permanently excluded, and the process is iterated. Backward elimination terminates when there is no feature whose exclusion leads to a lower error rate, or when the number of features reaches a specified level.

   Apply 3NN with backward elimination on this dataset with the minimum number of features being one. Which feature would be eliminated, if any?

3 Decision Trees [15 points]

1. *(7 points)* Consider the problem of predicting if a person has a college degree based on age and salary. The table and graph below contain training data for 10 individuals.

Age	Salary ($)	College Degree
24	40,000	Yes
53	52,000	No
23	25,000	No
25	77,000	Yes
32	48,000	Yes
52	110,000	Yes
22	38,000	Yes
43	44,000	No
52	27,000	No
48	65,000	Yes



Build a decision tree for classifying whether a person has a college degree by greedily choosing threshold splits that maximize information gain. What is the depth of your tree and the information gain at each split?

2. (4 points) A multivariate decision tree is a generalization of univariate decision trees, where more than one attribute can be used in the decision rule for each split. That is, splits need not be orthogonal to a feature's axis.

   For the same data, learn a multivariate decision tree where each decision rule is a linear classifier that makes decisions based on the sign of $\alpha x_{age} + \beta x_{income} - 1$.

   What is the depth of your tree, as well as α, β and the information gain for each split?

3. (4 points) Multivariate decision trees have practical advantages and disadvantages. List two advantages and two disadvantages multivariate decision trees have compared to univariate decision trees.

4 Programming Question [45 Points]

In this problem, you will train a logistic regression model to predict the Click Through Rate (CTR) on a dataset with about 10,000 examples. The CTR provides a measure of the popularity of an advertisement, and the features we will use for prediction include attributes of the ad and the user.

4.1 Unprocessed Dataset

The dataset we will consider comes from the 2012 KDD Cup Track 2. Here, a user types a query and a set of ads are displayed and we observe which ad was clicked.

For example:

1. Alice went to the famous search engine Elgoog, and typed the query "big data".
2. Besides the search result, Elgoog displayed 3 ads each with some short text including its title, description, etc.
3. Alice then clicked on the first advertisement

This completes a SESSION. At the end of this session Elgoog logged 3 records:

Clicked = 1 | Depth = 3 | Position = 1 | Alice | Text of Ad1 |
Clicked = 0 | Depth = 3 | Position = 2 | Alice | Text of Ad2 |
Clicked = 0 | Depth = 3 | Position = 3 | Alice | Text of Ad3 |

In addition, the log contains information about Alice's age and gender. Here is the format of a complete row of our training data:

Clicked | Depth | Position | Userid | Gender | Age | Text Tokens of Ad

Let's go through each field in detail:

- "Clicked" is either 0 or 1, indicating whether the ad is clicked.

- "Depth" takes a value in $\{1, 2, ..., \}$ specifying the number of ads displayed in the session.
- "Position" takes a value in $\{1, 2, ..., Depth\}$ specifying the rank of the ad among all the ads displayed in the session.
- "Userid" is an integer id of the user.
- "Age" takes a value in $\{1, 2, 3, 4, 5, 6\}$, indicating different ranges of a user's age: ''1' for (0, 12], '2' for (12, 18], '3' for (18, 24], '4' for (24, 30], '5' for (30, 40], and '6' for greater than 40.
- "Gender" takes a value in $\{-1, 1\}$, where -1 stands for male and 1 stands for female.
- "Text Tokens" is a comma separated list of token ids. For example: "15,251,599" means "token_15", "token_251", and "token_599". (Note that due to privacy issues, the mapping from token ids to words is not revealed to us in this dataset, e.g., "token_32" to "big".)

Here is an example that illustrates the concept of features "Depth" and "Position". Suppose the list below was returned by Elgoog as a response to Alice's query. The list has *depth* = 3. "Big Data" has *position* = 1, "Machine Learning" has *position* = 2 and so forth.

Big Data
Machine Learning
Cloud Computing

Here is a sample from the training data:
0 | 2 | 2 | 280151 | 1 | 2 | 0, 1, 154, 173, 183, 188, 214, 234, 26, 3, 32, 36, 37, 4503, 51, 679, 7740, 8, 94
The test data is in the same format except that each instance does not have the first label field, which is stored in a separate file named "test_label.txt". Some data points do not have user information. In these cases, the userid, age, and gender are set to zero.

4.2 Processed Dataset (that you will use...)

In class, we simply denote

$$x^t = [x_1^t, ..., x_d^t] \tag{1}$$

as an abstract feature vector. In the real world, however, constructing the feature vector requires some thought. We have preprocessed the dataset for you, making the following modifications.

- First of all, not everything in the data should be treated as a feature. In this dataset, "Userid" should not be treated as feature. We have removed this column from the dataset for you.
- Similarly, we cannot directly use the list of token ids as features in Equation **??** since the numbers are arbitrarily assigned and thus meaningless for the purposes of regression. Instead, we should think of the list of token ids $L = [l_1, l_2, l_3, ...]$ as a compact representation of a sparse binary vector b where $b[i] = 1 \forall i \in L$. We have replaced the list of words with a list of binary variables indicating token presence.

- As for the rest of the features: “Depth”, “Position”, “Age”, and ‘Gender”, they are scalar variables, so we maintain their original values in the dataset.

The dataset that we are giving you has the following form:
Clicked, Depth, Position, Gender, Age, Word1, Word2, Word3, Word4, ..., Word50

So a sample training datapoint would be:
0, 1, 2, 1, 12, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0

Again, the test data is in the same format except that each instance does not have the first label field, which is stored in a separate file named “test_label.txt”.

4.3 Accessing and Processing the Data

1. Download the processed version of the dataset “clickprediction_data.tar.gz” from the course website.

2. After extracting, there should be five files: train.txt, test.txt test_label.txt, start.py (which contains code to load the data) and oversampled_train.txt (more on this file later. for now, just ignore it). I already added a column of ones in the starter code, so if you don’t want to do that you can just remove the add_ones function.

4.4 Batch Gradient Descent

Recall that logistic regression can be optimized by gradient descent (or ascent). I’m calling it batch gradient descent because we’re using the whole training data in order to update the weights in every iteration, in contrast to stochastic gradient descent. The log loss function you should use is the following:

$$-((\sum_{j=1}^{N} y^j(w_0 + \sum w_i x_i^j) - ln(1 + e^{w_0 + \sum w_i x_i^j}))/N - \frac{\lambda}{2}||w||_2^2) \tag{2}$$

1. Write down the equation for the weight update step. That is, how to update weights w^{t+1} using (X, Y, w^t). Assume we’re doing L2 regularization.

2. Implement gradient descent and run it for 1000 iterations, with different stepsizes η and λ parameters. Initialize w with a vector of zeroes. Remember not to regularize w_0!

   (a) Plot the log-loss function after each iteration (no need to plot it for iteration 0), using $\eta = 0.1$ and $\lambda = 0.3$. Remember to include the regularization term.

   (b) Use the weights you found to predict the CTRs for the test data. Recall that “test_label.txt” contains the labels for the test data. Report the SSE (sum of squared errors) of your predicted CTR. Make sure to use the SSE for the 0/1 prediction (meaning that your prediction should be either 0 or 1, and not probability value. Make prediction be 1 if the probability if class 1 is greater than 0.5). Report the SSE for the same parameters you used in the previous question.

3. Now implement the following stop criteria: stop when $|l(w^t) - l(w^{t-1})| < \epsilon$, where l is the log-loss function. Run gradient descent with $\eta = 0.1$, $\lambda = 0.3$ and $\epsilon = 0.0005$.
   (a) For how many iterations did gradient descent run?
   (b) Plot the log-loss function after each iteration for these iterations (now the x axis should not go until 1000 as it did in the previous question).
   (c) Use the weights you found to predict the CTRs for the test data. Report the SSE (sum of squared errors) of your predicted CTR.

4.5 Stochastic Gradient Descent

Recall that stochastic gradient descent (SGD) performs a gradient descent using a noisy estimate of the full gradient based on just the current example.

1. Write down the equation for the weight update step. That is, how to update weights w^{t+1} using the data point (x^j, y^j) and w^t, where $x^j = [x_1^j, x_2^j, ..., x_d^j]$ is the feature vector for example j, and $y^j \in \{0, 1\}$ is the label. Once again, use L2 regularization.
2. implement SGD and train the weights by making one pass over the dataset. You may assume that the data is randomly ordered, so don't reorder it. Use $\eta = 0.1$.
   (a) Report the l_2 norm of the weights at the end of the pass with $\lambda = 0$ (no regularization). Note that if your l_2 norm is not small and keeps growing as you get more and more data, you may be using a λ too small. Report the same norm for $\lambda = 0.3$. Please do not include w_0 when calculating the l_2 norm, as it is not regularized.
   (b) Use the weights you found with $\lambda = 0.3$ to predict the CTRs for the test data and report the SSE.
   (c) Report the weight for the following features: intercept (w_0), depth and position. Use $\lambda = 0.3$.
3. Now run SGD for 5 passes over the data, with $\eta = 0.1$ and $\lambda = 0.3$. Report the value of the log-loss function on the training data, after the 5 passes. Run batch gradient descent for 5 iterations (the computing cost will be in the same ballpark) and the same parameters, and report the value of the log-loss function. Which algorithm seems to converge faster?

4.6 Class Imbalance

Now take a look at your predictions on the test data, for either SGD or batch gradient descent. More specifically, take a look at how many times your algorithm predicted a 1. Due to class imbalance, SSE is not an adequate measure of quality here. There is a hidden lesson here: you should never trust an evaluation metric blindly, without thinking about what it means and even looking at some predictions or at the model. We will now use two class-specific metrics: precision and recall, defined as follows.

Given a class c, lets say that there are two groups: A, the group of instances that are in class c, and B, the group of instances that your algorithm classified as being in class c. Precision is defined as $\frac{|A \cap B|}{|B|}$, and recall is defined as $\frac{|A \cap B|}{|A|}$. If an algorithm doesn't classify any instances on class c, let's define the precision as 0.

1. Calculate precision and recall for classes 0 and 1 on the test set, for the predictions made by SGD running with one pass over the data, $\lambda = 0.3$ and $\eta = 0.1$.

2. A very simple (and not the best!) solution to the problem of class imbalance is to oversample from the class that has fewer instances until the imbalance has been corrected. This introduces other biases into the dataset, as the instances of that class will be repeated over and over again. The file with a new training set with more balanced data (where I oversampled the positive examples) is "training_oversampled.txt". Now run the batch gradient descent with 10000 iterations, $\eta = 0.01$, $\lambda = 0.3$) on this training set and report the precision/recall values for classes 0 and 1 for the test data.