CSE 446 Machine Learning, Winter 2015
Homework 2

Due: Fri, Feb 13, 9:30am

1 Naive Bayes with discrete features [25 Points]

Consider the following data set on lung diseases. Your goal is to build a Naive Bayes classifier that predicts whether a person have Bronchitis or Tuberculosis, given his/her symptoms.

Disease	X-ray Shadow	Dyspnea	Lung inflammation
Bronchitis	Yes	Yes	Yes
Bronchitis	Yes	Yes	Yes
Bronchitis	No	No	Yes
Tuberculosis	No	No	Yes
Tuberculosis	Yes	Yes	No
Tuberculosis	Yes	No	No

1. *(10 Points)* List the distributions that would be learned if you use MLE to estimate a Naive Bayes model from this data (e.g. $P(Dyspenea|Bronchitis)$). Include all of the estimated probabilities for each distribution. Show your work.

2. *(10 Points)* Based on your learned Naive Bayes model, diagnose a patient with the following symptoms. Show your work.

X-ray Shadow	Dyspnea	Lung inflammation
Yes	No	Yes

3. *(5 Points)* Describe briefly wether you think your learned model is overfit. List one key sign of overfitting in Naive Bayes models, as described in class. Does your model have this problem? Why or why not?

2 Logistic Regression Programming Question [40 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.

2.1 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:
$0 \mid 2 \mid 2 \mid 280151 \mid 1 \mid 2 \mid 0, 1, 154, 173, 183, 188, 214, 234, 26, 3, 32, 36, 37, 4503, 51, 679, 7740, 8, 94$
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.

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

In class, we simply denote

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

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 a 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 1 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, 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".

2.3 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).

2.4 Batch Gradient Descent (30 points)

Recall that logistic regression can be optimized by gradient descent (or ascent). We call 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. In class you discussed maximizing the regularized conditional log-probability. Here we'll be minimizing the regularized log-loss (the updates will be the same):

$$l(\mathbf{x}, w_0, \mathbf{w}) = \frac{1}{2}\lambda||\mathbf{w}||_2^2 - \frac{1}{N}\ln\prod_j P(y^j|\mathbf{x}^j, w_0, \mathbf{w})$$

1. *(3 pts)* Write down the equation for the batch weight update step. That is, how to update weights w^{t+1} using $(\mathbf{X}, \mathbf{y}, w^t, \lambda)$. Assume we're using l^2 regularization and step size η.

2. *(9 pts)* Implement gradient descent and run it for 1000 iterations, initializing w with a vector of all zeros. Remember not to regularize w_0!

   (a) Plot the log-loss function after each iteration, using $\eta = 0.1$ and $\lambda = 0.3$. Remember to include the regularization term.

   (b) Using the weights learned above, predict the CTRs for the test data and report the accuracy of prediction.

3. *(9 pts)* 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 (now the x-axis should not go until 1000 as it did in the previous question).

   (c) Use the weights you learned to predict the CTRs for the test data, and report the accuracy that is achieved.

4. *(9 pts)* Finally, we will experiment with the effect of λ on the magnitudes of weights. Run batch gradient descent for 1000 iterations on the data with $\lambda = 0$, and report the l^2 norm of the weights at the end (excluding the offset w_0). Repeat with $\lambda = 0.3$. Use $\eta = 0.1$. Briefly state which value is preferred and why.

2.5 Class Imbalance (10 points)

A training data set with binary labels is said to be imbalanced, if one of the labels is underrepresented compared to the other label. The CTR data that we are currently working with has this issue. In this section, we will look at a simple way to resolve this.

You should also familiarize yourself with two new metrics for prediction quality: precision and recall. Precision is defined as the number of true positives divided by the total number of elements labeled as belonging to the positive class. Recall is defined as the number of true positives divided by the total number of elements that actually belong to the positive class. In other words, precision measures the exactness of your prediction, whereas recall measures the completeness of your prediction.

1. *(5 pts)* Run 5000 iterations of batch descent over the data with $\lambda = 0.3$ and $\eta = 0.01$. Using the learned weights, report the precision and recall for classes 0 and 1 on the test set.

2. *(5 pts)* 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 several times. The file "training_oversample.txt" contains a more even distribution of classes (the positive examples are oversampled). Now run 5000 iterations of batch descent on this training set ($\lambda = 0.3, \eta = 0.01$), and report the precision/recall values for classes 0 and 1 on the test set.

3 Perceptron Programming Question [30 Points]

In this problem, we will reuse the data from the last problem, but instead train a Perceptron. This problem is more open ended, you should follow the algorithm as described in class but it is up to you to decide how to set all of the appropriate hyperparameters and otherwise demonstrate how to do good experimental design.

1. *(10 pts)* Report test precision and recall for your Perceptron when trained on the "training_oversample.txt" data from above. Briefly compare its performance to the logistic regression model, justifying any differences you see.

2. *(10 pts)* Describe your experimental setup, including any hyperparameters you used, wether you had a dev set or cross validation, etc. There is no single correct answer here, but we want to see that you don't ignore any of the key issues we described in class.

3. *(5 pts)* Does your perceptron converge when run on the "training_oversample.txt" data? Briefly justify why or why not.

4. *(5 pts)* If you were faced with a new problem, it is interesting to consider wether you might try logistic regression of Perceptron first. Briefly state one advantage of each approach (possible answers include differences in runtime, accuracy, complexity of implementation, etc.). Which approach is your favorite and why? (the last question has no wrong answer, but for this assignment you must pick one!)