CS 475 Machine Learning: Homework 3
Generalized Linear Models
Due: Wednesday October 10, 2012, 12pm
100 Points Total Version 1.0

1 Programming (50 points)

In this assignment you will write two learning algorithms: a Naive Bayes classifier, and a Perceptron classifier. Both classifiers need only support binary prediction.

1.1 Naive Bayes

Implement a Naive Bayes classifier using the maximum likelihood solutions presented in class. A few details will help your implementation:

Continuous features The Naive Bayes equations naturally handle binary features. For continuous features, select the mean of the feature *in the entire training set*. If a feature's value is *less than or equal to* the mean, pretend as if feature A is observed. If the feature's value is *greater than* the mean, pretend as if feature B is observed. Therefore, each continuous feature will be split into two features (A and B). Remember, for computing the mean value of a feature, examples in which the feature does not appear have value 0 for that feature. Remember, you can determine binary features by examining features in the data that only appear with value 1 or 0.

New Features There may be new features encountered at test time that are not present in the training data. For simplicity, skip these features when creating predictions.

Note that there is no bias term in this version and you should *not* include one in your solution.Implement this algorithm as the `NaiveBayesPredictor` class. Your Naive Bayes classifier should be selected by passing the string `naive_bayes` as the argument for `algorithm.`

1.1.1 Smoothing

Naive Bayes is prone to overfitting because test instances with rare or unobserved features could be assigned unfairly low probabilities for some or all labels. Therefore, you will *smooth* your probability estimates using a technique commonly called add-λ smoothing.

Smoothing these distributions is simple: increase each feature/label count by λ, for some real-valued $\lambda \geq 0$. For this reason, the λ terms are often referred to as *pseudocounts*, because you are pretending to count each feature/label more than actually observed. This makes it so that unseen features will still have some probability mass at test time. When $\lambda = 1$, this is called Laplacian smoothing (or simply +1 smoothing).

Remember to adjust the normalization constants accordingly so that your probabilities still sum to 1. You should smooth both $p(x|y)$ and $p(y)$.

You *must* add a command line argument to allow λ to be adjusted via the command line. The default value should be 1.0. Add this command line option by adding the following code to the `createCommandLineOptions` method of `Classify`.

```
registerOption("lambda", "double", true, "The level of smoothing for Naive Bayes.");
```

You can then read the value from the command line by adding the following to the main method of `Classify`:

```
double lambda = 1.0;
if (CommandLineUtilities.hasArg("lambda"))
    lambda = CommandLineUtilities.getOptionValueAsFloat("lambda");
```

1.1.2 Making Predictions

When making predictions using the learned Naive Bayes model, you should use the following prediction rule: choose the label y which satisfies

$$\arg \max_{y \in \{-1,1\}} p(y) \prod_{j \in \mathbf{x}} p(x_j|y)$$

where the product is over the features contained in the instance $\mathbf{x}$. In the case of a tie, let $y = 1$.

Handling Underflow Notice that the prediction rule involves a product of multiple terms which may have values close to zero. If you repeatedly multiply small values together, this product will quickly shrink, and the floating point is likely to underflow. The most common way to deal with this issue is to convert the product into a summation by taking the log of the probabilities. Recall that $\arg\max_x f(x) = \arg\max_x \log f(x)$, because $\log(x)$ monotonically increases with x. As you learned in class, this property is also exploited in maximum likelihood estimation, where the goal is to maximize the log-likelihood, because it is analytically easier to work with log probabilities. The prediction rule you should use is thus:

$$\arg \max_{y \in \{-1,1\}} \log(p(y)) + \sum_{j \in \mathbf{x}} \log(p(x_j|y))$$

1.1.3 Running Code

We will evaluate your algorithm by running the following commands:

```
java cs475.Classify -mode train -algorithm naive_bayes -model_file speech.naive_bayes.model \
                    -data speech.train
```

To run the trained model on development data:

```
java cs475.Classify -mode test -model_file speech.naive_bayes.model -data speech.dev \
                    -predictions_file speech.dev.naive_bayes.predictions
```

1.2 Linear Threshold Classification: Perceptron and Winnow

Perceptron and Winnow are mistake-driven online learning algorithms for linear classifiers. They take as input a vector of real-valued inputs $\mathbf{x}$ and make a prediction $\hat{y} \in \{-1, +1\}$ (for this assignment we consider only binary labels). Predictions are made using a linear classifier known as a linear threshold unit (LTU): $\hat{y} = \text{sign}((\mathbf{w} \cdot \mathbf{x}) - \beta)$, with a scalar threshold β. The term $\mathbf{w} \cdot \mathbf{x}$ is the dot product of $\mathbf{w}$ and $\mathbf{x}$ computed as $\sum_i x_i w_i$. An LTU says that if this dot product is greater than the threshold β then the prediction is $+1$; if the dot product is less than the threshold β, the prediction is -1.

Updates to $\mathbf{w}$ are made only when a prediction is incorrect: $\hat{y} \neq y$. The new weight vector $\mathbf{w}'$ is a function of the current weight vector $\mathbf{w}$ and example $(\mathbf{x}, y)$. The weight vector is updated so as to improve the prediction on the current example. The only difference between Perceptron and Winnow is the update rule: Perceptron uses additive updates, while Winnow uses multiplicative updates (explained below). Note that these algorithms naturally handle both continuous and binary features, so no special processing is needed.

1.2.1 Basic Algorithm

The basic structure of the linear threshold based algorithms is:

1. Initialize $\mathbf{w}$, set learning rate η, threshold β, and number of iterations I
2. For each training iteration $k = 1 \ldots I$:
   (a) For each example $i = 1 \ldots N$:
      i. Receive an example $\mathbf{x}_i$
      ii. Predict the label $\hat{y}_i = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x}_i \geq \beta \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{x}_i < \beta \end{cases}$
      iii. If $\hat{y}_i \neq y_i$, make an update to $\mathbf{w}$. The update $\mathbf{w}'$ depends on the specific algorithm (described below).

1.2.2 Thick Separators

The basic linear threshold algorithm tries to learn weights $\mathbf{w}$ so that $\mathbf{w} \cdot \mathbf{x}$ is always on the correct side of the threshold β. It can be desirable to enforce a *thick separator* during training. In this setting, not only do we want $\mathbf{w} \cdot \mathbf{x}$ to be on the correct side of β, but we want it to be far away from β: sufficiently larger than β for positive labels, and sufficiently smaller than β for negative labels. This can reduce the chance of overfitting, because the algorithm will learn a separator that is far from all of the training instances – this is related to the principle of max-margin classification.

The prediction rule in step 2(a)ii should be replaced with:

$$\hat{y}_i = \begin{cases} 1 \text{ if } \mathbf{w} \cdot \mathbf{x}_i \geq \beta + \tau \\ -1 \text{ if } \mathbf{w} \cdot \mathbf{x}_i \leq \beta - \tau \\ 0 \text{ otherwise} \end{cases}$$

This rule uses a user-supplied thickness parameter $\tau \geq 0$. This is equivalent to the basic algorithm in section 1.2.1 when $\tau = 0$.

If $\hat{y}_i = 0$, it is considered an incorrect label, because $y_i \in \{-1, +1\}$. This means that if the value of $\mathbf{w} \cdot \mathbf{x}_i$ falls within the bounds of the separator $(-\tau, \tau)$, it will be treated as a misclassification.

You *must* add a command line argument to allow τ to be adjusted via the command line. The default value should be 0. Add this command line option by adding the following code to the `createCommandLineOptions` method of `Classify`.

```
registerOption("thickness", "double", true, "The value of the linear separator thickness.");
```

You can then read the value from the command line by adding the following to the main method of `Classify`:

```
double thickness = 0.0;
if (CommandLineUtilities.hasArg("thickness"))
    thickness = CommandLineUtilities.getOptionValueAsFloat("thickness");
```

1.2.3 Perceptron

The Perceptron algorithm is based on *additive* updates to train the weight vector $\mathbf{w}$. The update rule (step 2(a)iii) for Perceptron is: $\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$.

The threshold β should be set to 0 for Perceptron. The weights should be initialized to 0: $\mathbf{w} = \mathbf{0}$.

Implement this algorithm as the `PerceptronPredictor` class. Note that there is no bias term in this version and you should *not* include one in your solution. Your Perceptron predictor will be selected by passing the string `perceptron` as the argument for the algorithm parameter.

1.2.4 Winnow

The Winnow algorithm is based on *multiplicative* updates to train the weight vector $\mathbf{w}$. The update rule (step 2(a)iii) for Winnow is: $\mathbf{w}' = \mathbf{w} \times \eta^{y_i s_i}$, where $s_i = 1$ if $\mathbf{x}_i \geq 0$ and $s_i = -1$ if $\mathbf{x}_i < 0$. That is, $s_i = \text{sign}(\mathbf{x}_i)$.

The threshold β should be set to $\frac{N}{2}$ for Winnow, where N is the number of training instances.

The weights should be initialized to 1: $\mathbf{w} = \mathbf{1}$.

Implement this algorithm as the `WinnowPredictor` class. Note that there is no bias term in this version and you should *not* include one in your solution. Your Winnow predictor will be selected by passing the string `winnow` as the argument for the algorithm parameter.

Handling Overflow As the algorithm proceeds, it is possible that the weights for certain features will repeatedly be multiplied by η. This could blow up: if a weight is increased too many times, eventually the floating point will overflow. We will ask you to handle this by taking a simple approach of cropping the value of a weight if it gets too high. That is, if you ever make an update $\mathbf{w}'$ which is higher than some upper bound μ, then you should simply set $\mathbf{w}' = \mu$. This will prevent the weight from becoming arbitrarily large. The upper bound μ should be set to: `1.0e6`.[1]

1.2.5 Learning Rate

Perceptron and Winnow use a learning rate η, where $0 < \eta \leq 1$ for Perceptron and $1 < \eta \leq 2$ for Winnow. Your default value for η should be 1 for Perceptron and 2 for Winnow. You *must* add a command line argument to allow this value to be adjusted via the command line.

Add this command line option by adding the following code to the `createCommandLineOptions` method of `Classify`.

```
registerOption("online_learning_rate", "double", true, "The LTU learning rate.");
```

Be sure to add the option name exactly as it appears above. A common mistake is to change underscores to dashes.

You can then read the value from the command line by adding the following to the main method of `Classify`:

```
double online_learning_rate = algorithm.equals("winnow") ? 2.0 : 1.0;
if (CommandLineUtilities.hasArg("online_learning_rate"))
    online_learning_rate = CommandLineUtilities.getOptionValueAsFloat("online_learning_rate");
```

[1] You should be aware that it is also possible to handle overflow by working with the log of the weights rather than the weights directly, in the same way that you work with log probabilities to avoid underflow in Naive Bayes. However, if you work with weights in log space, then you must also perform predictions $\text{sign}((\mathbf{w} \cdot \mathbf{x}) - \beta)$ in log space. In addition to taking the log of products (which simply becomes a sum), this also involves working with the log of sums, which is not as straightforward. We are instead asking you to simply crop the weights, since it is trickier to implement Winnow in log space.

1.2.6 Number of training iterations

Since we will be running these online methods in the batch setting, you can iterate multiple times over the data. This can improve performance by increasing the number of updates each algorithm makes. We will define the number of times each algorithm iterates over the data by the parameter `online_training_iterations`. You *must* define a new command line option for this parameter. Use a default value of 1 for this parameter.

You can add this option by adding the following code to the `createCommandLineOptions` method of `Classify`.

```
registerOption("online_training_iterations", "int", true, "The number of training iterations for LTU.");
```

You can then read the value from the command line by adding the following to the main method of `Classify`:

```
int online_training_iterations = 1;
if (CommandLineUtilities.hasArg("online_training_iterations"))
    online_training_iterations = CommandLineUtilities.getOptionValueAsInt("online_training_iterations");
```

1.3 Implementation Details

Because Perceptron and Winnow are so similar, you could end up duplicating code unnecessarily. We recommend implementing an *abstract* class called `LinearThresholdPredictor`, which implements the general algorithm described in sections 1.2.1 and 1.2.2. The classes `PerceptronPredictor` and `WinnowPredictor` would extend this class to fill in the specific update rules and β values.

For all numerical calculations involving floating point numbers, use the `double` type and NOT the `float` type to store values. This will help in achieving numerical precision.

2 Analytical (50 points)

1) Basic Concepts (20 points) Generalized linear models (GLMs), especially logistic regression are heavily used by banks, credit card companies and insurance companies. Actually, when you apply for a credit card, banks may put your information into a logistic regression model to decide whether you are eligible.

(a) The GLMs are closely related to the exponential distribution family, which has the probability density/mass function $f(X = x; \theta)$ in the form

$$f(X = x; \theta) = h(x)e^{\eta(\theta)\cdot T(x) - A(\theta)}, \tag{1}$$

where h, η, T, A are some known functions. Given

(1) the probability mass function of the binomial distribution (assume that n is given)

$$f(X = x; p) = \binom{n}{x} p^x (1-p)^{n-x}, \ x \in \{0, 1, ..., n\}; \tag{2}$$

(2) the probability mass function of the Poisson distribution

$$f(X = x; \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}; \tag{3}$$

(3) the probability density function of the Gaussian distribution (assume that σ is given)

$$f(X = x; \mu) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4}$$

Please show that these three distributions belong to the exponential family.

(b) We see that $\eta(\theta)$ for the binomial distribution is exactly the log-odds-ratio in the logistic regression. That is why that the logistic regression is also called binomial regression. Therefore given $\eta(\theta)$ for the Poisson distribution, and n observations $(x_i, y_i)_{i=1}^n$, what is the log-likelihood function for the Poisson regression?

(c) The GLMs often contain some transformation, which is non-linear such as the log-odds-ratio transformation in the logistic regression. Why do we still call them "linear"?

(d) For logistic regression, if the data are linear separable, please show that the maximum of the log-likelihood function is $+\infty$. By imposing ℓ_1 or square norm constraint on β, we can make the maximum likelihood estimation feasible. Why?

2) Naive Bayes v.s. Logistic Regression (10 points) People use naive Bayes and logistic regression as illustrative examples to compare generative and discriminative models.

(a) In a classification problem, we assume that the data come from a mixture of two d-dimensional Gaussian distributions (each Gaussian distribution represents a class). How many and what parameters are used to describe such a mixture?

(b) How many parameters are used in logistic regression? How many parameters are used in naive Bayes? What assumption does naive Bayes make to reduce the number of the parameters?

(c) What is the difference between generative and discriminative models in terms of their likelihood functions?

3) Missing Values (10 points) One advantage of naive Bayes over the logistic regression is that it can handle missing values in the data (i.e. not knowing the value of a specific feature). Given a trained naive Bayes classifier, show how to calculate $\mathbb{P}(Y, X_1, X_2, ..., X_{d-1})$ where X_d is unknown. Assume $Y \in \{0, 1\}$ and $X_j \in \{1, 2, ..., K\}$ where $j = 1, ..., d$. [Hint: The conditional probability in naive Bayes can be completely factorized.]

4) Add-λ Smoothing (10 points) When training a naive Bayes classifier, it is possible to assign zero values to some conditional probabilities. Given n observations, $(x_i, y_i)_{i=1}^n$, where $x_i = (x_{i1}, ..., x_{id})$, we then have

$$\hat{\mathbb{P}}(X_j = x|Y = y) = \frac{\sum_{i=1}^n I(x_{ij} = x, y_i = y)}{\sum_{i=1}^n I(y_i = y)}. \tag{5}$$

where $I(x_{ij} = x, y_i = y)$ is the indicator function that is 1 iff the jth feature in the ith example has value x and the label for the ith example is y.

For example, if $\sum_{i=1}^n I(x_{ij} = 1, y_i = 0) = 0$, then $\hat{\mathbb{P}}(X_j = x|Y = y) = 0$. While some people prefer to use a different estimator

$$\hat{\mathbb{P}}(X_j = x|Y = y) = \frac{\lambda + \sum_{i=1}^n I(x_{ij} = x, y_i = y)}{K\lambda + \sum_{i=1}^n I(y_i = y)}. \tag{6}$$

(6) is is so-called "add-λ smoothing" procedure. What role does λ play in terms of the bias variance tradeoff?

3 What to Submit

In each assignment you will submit two things.

1. **Code:** Your code as a zip file named `library.zip`. **You must submit source code (.java files)**. We will run your code using the exact command lines described above, so make sure it works ahead of time. Remember to submit all of the source code, including what we have provided to you.

2. **Writeup:** Your writeup as a **PDF file** (compiled from latex) containing answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and use the provided latex template for your answers.

Make sure you name each of the files exactly as specified (library.zip and writeup.pdf).

To submit your assignment, visit the "Homework" section of the website (http://www.cs475.org/.)

4 Questions?

Remember to submit questions about the assignment to the appropriate group on the class discussion board: http://bb.cs475.org.