CS 4269/6362 Machine Learning: Homework 2
Linear Models for Supervised Learning

Due: 1/28/2016

150 Points Total Version 1.0

1 Programming (100 points; 150 points for CS 4269)

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

1.1 Linear Regression

Recall from the lecture that linear regression model takes the form

$$f(x) = w^T x,$$

where x is a feature vector, and we assume that $x_1 = 1$ (this unit feature can always be appended to the given feature vector; indeed, you will append this unit feature in your implementation of the linear regression). Given a data set of examples where $\mathbf{X}$ is a matrix with rows as feature vectors corresponding to different data points and y is a corresponding observed output vector, we can train the weight vector w as follows:

$$w = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y.$$

Your task is to implement linear regression training and prediction.

While you will be allowed to use the Apache math commons package for this assignment, **you can only use the libraries in org.apache.commons.math3.linear** (i.e., the linear algebra libraries).

1.1.1 Regression Data

For this portion of the assignment you will use the provided regression dataset.

NASDAQ A popular example of regression is to predict stock prices. The goal of this dataset is to predict the price of the NASDAQ 500 index each day given the index value from the three previous days. The three attributes correspond to the price the day before

(1), two days prior (2) and three days prior (3). We leave it to you to determine what reasonable accuracy for this data set should be.

Note that the script `compute_accuracy.py` does not support regression. Instead, we are providing a new script `compute_regression_accuracy.py` which computes the mean squared error. Note that the output says Accuracy, but it's not. Take a look at the Python code if you are curious.

1.1.2 Running Code

We will evaluate your algorithm by running the following commands for **easy**, **hard**, and **nasdaq** data sets:

```
java cs362.Learn -mode train -algorithm linear_regression -model_file easy.linear_regression.model \
                    -data easy.train - task regression
```

To run the trained model on development data:

```
java cs362.Learn -mode test -model_file easy.linear_regression.model -data easy.dev \
                    -predictions_file easy.dev.linear_regression.predictions -task regression
```

1.2 Naive Bayes

Implement a Naive Bayes classifier presented in class. A few details will help your implementation:

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 `NaiveBayesClassifier` class. Your Naive Bayes classifier should be selected by passing the string `naive_bayes` as the argument for `algorithm`.

1.2.1 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 `Learn`.

```
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 `Learn`:

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

1.2.2 Handling Non-Binary Features

In your code you will treat all features as binary. For all features which are not binary (e.g., continuous features), assume that the feature value is 0 when it is < 0.5 and 1 otherwise.

1.2.3 Making Predictions

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

$$\arg\max_{y \in \{0,1\}} p(y) \prod_{j \in \mathbf{x}} p(x_j|y)$$

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:

$$\arg\max_{y \in \{0,1\}} \log(p(y)) + \sum_{j \in \mathbf{x}} \log(p(x_j|y))$$

1.2.4 Running Code

We will evaluate your algorithm by running the following commands (we'll additionally specify the required command line parameter):

```
java cs362.Learn -mode train -algorithm naive_bayes -model_file bio.naive_bayes.model \
                    -data bio.train -task classification
```

To run the trained model on development data:

```
java cs362.Learn -mode test -model_file bio.naive_bayes.model -data bio.dev \
                    -predictions_file bio.dev.naive_bayes.predictions -task classification
```

1.2.5 Classification Data

For this portion of the assignment you will use the bio, easy, and hard data sets from homework 1.

1.3 Perceptron Algorithm

Perceptron is a mistake-driven online learning algorithms for linear classifiers. It takes 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. Below, we will use the threshold $\beta = 0$. **BEWARE: because the data sets we provide code output values as 0/1 rather than -1/+1, you will need to convert the observed labels to the -1/+1 coding scheme, and vice versa as you train and evaluate the model.**

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. Note that this algorithm naturally handles both continuous and binary features, so no special processing is needed.

1.3.1 Basic Algorithm

The basic structure of the linear threshold based algorithms is:

1. Initialize $\mathbf{w} = \mathbf{0}$, set learning rate η 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 $\hat{y_i} = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x}_i \geq 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{x}_i < 0 \end{cases}$
      iii. If $\hat{y_i} \neq y_i$, make an update to $\mathbf{w}$. The update $\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$.

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

1.3.2 Learning Rate

Perceptron uses a learning rate η, where $0 < \eta \leq 1$. Your default value for η should be 1. 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 `Learn`.

```
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 `Learn`:

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

1.3.3 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 `Learn`.

```
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 `Learn`:

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

1.3.4 Running Code

We will evaluate your algorithm by running the following commands (we'll additionally specify the required command line parameters):

```
java cs362.Learn -mode train -algorithm perceptron -model_file bio.perceptron.model \
                    -data bio.train -task classification
```

To run the trained model on development data:

```
java cs362.Learn -mode test -model_file bio.perceptron.model -data bio.dev \
                    -predictions_file bio.dev.perceptron.predictions -task classification
```

1.3.5 Classification Data

For this portion of the assignment you will use the bio, easy, and hard data sets from homework 1.

1.4 Implementation Details

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 CS 6362 ONLY: Analytical (50 points)

NOTE: PLEASE USE THE PROVIDED answers_form.tex TEMPLATE FOR YOUR ANSWSERS! SUBMIT THE COMPILED PDF.

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) When the probability of an event, such as a coin landing heads or classifier output being $y = +1$, is p, the odds ratio is $p/(1-p)$. 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"?

2) 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.]

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

4) Stochastic Gradient Algorithm (10 points) The stochastic gradient algorithm is a very powerful optimization tool to solve many online learning problems. Instead of computing the gradient over the entire data set before making an update, the stochastic gradient algorithm computes the gradient over a single example, then updates the parameters. By passing over the entire data set in this fashion we can converge to the optimal parameters.

A single iteration of stochastic gradient considers a single example. As a result, we know that stochastic gradient converges more slowly than gradient descent. While it takes many more iterations, each iteration is much faster, both in terms of memory and computation.

Consider a linear regression problem with n samples:

$$\hat{\beta} = \arg\min_{w} \frac{1}{2} \sum_{i=1}^{n} (y_i - w^T x_i)^2. \tag{7}$$

In each iteration, instead of using only one example, we randomly choose k out of n samples and obtain $(x_{1'}, y_{1'}), ..., (x_{k'}, y_{k'})$.

(a) What is the computational complexity of computing the gradient at each iteration (using k and n)?

(b) What are the advantages/disadvantages of increasing k in terms of computational complexity (using k and n)? What is traded-off by increasing/decreasing k?

(c) What is an example of an algorithm we learned in class that uses stochastic gradient? Explain.

3 What to Submit

In each assignment you will submit two things on OAK.

1. **Code:** Your code as a zip file named `hw2code.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 (`hw2solution.pdf`) 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 (`hw2code.zip` and `hw2solution.pdf`).