CS 475 Machine Learning: Homework 2
Supervised Classifiers 1,
Probability, Linear Algebra and Decision Trees

Due: Tuesday September 29, 2015, 11:59pm

100 Points Total Version 1.0

Make sure to read from start to finish before beginning the assignment.

1 Programming (50 points)

In this assignment you will write a logistic regression classifier. Your implementation will be very similar to the algorithm we covered in class. Your code needs to handle data with binary and continuous features and only binary labels (no multi-class).

1.1 Logistic Regression

The logistic regression model is used to model binary classification data. Logistic regression is a special case of generalized linear regression where the labels Y are modeled as a linear combination of the data X, but in a transformed space specified by g, sometimes called the "link function":

$$E[y \mid \mathbf{x}] = g(\mathbf{w}^T \mathbf{x}). \tag{1}$$

This "link function" allows you to model inherently non-linear data with a linear model. In the case of logistic regression, the link function is the logistic function:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

1.2 Stochastic Gradient Descent

In this assignment, we will solve for the parameters $\mathbf{w}$ in our logistic regression model using stochastic gradient descent to find the maximum likelihood estimate.

Stochastic gradient descent (SGD) is an optimization technique that is both very simple and powerful. Regular gradient descent works by taking the gradient of the objective function and taking steps in directions where the gradient is negative, which decreases the objective function[1]. In stochastic gradient descent instead of exactly finding the gradient of the objective function, which is the expectation of the gradient over all of your training examples, an estimate is taken by sampling from the data. Evaluating the gradient function at a sample of the data produces a random value (the "stochastic" part of SGD) that has the same expectation as the true gradient, but higher variance. It turns out that using this stochastic gradient instead of the true gradient can often speed up optimization. In this assignment we will sample our data by only using one training instance, $(y, \mathbf{x})$, as is most common in SGD.

[1] Gradient descent decreases the objective function if the gradient (first order approximation) is locally accurate.

1.3 Maximum Conditional Likelihood

Since we seek to maximize the objective, we will use stochastic gradient *ascent*. We begin by writing the conditional likelihood:

$$P(Y \mid \mathbf{w}, X) = \prod_{i=1}^{n} p(y_i \mid \mathbf{w}, \mathbf{x}_i) \tag{3}$$

Since $y_i \in \{0, 1\}$, we can write the conditional probability inside the product as:

$$P(Y \mid \mathbf{w}, X) = \prod_{i=1}^{n} p(y_i = 1 \mid \mathbf{w}, \mathbf{x}_i)^{y_i} \times (p(y_i = 0 \mid \mathbf{w}, \mathbf{x}_i))^{1-y_i} \tag{4}$$

Note that one of these terms in the product will have an exponent of 0, and will evaluate to 1.

For ease of math and computation, we will take the log:

$$\ell(Y, X, \mathbf{w}) = \log P(Y \mid \mathbf{w}, X) = \sum_{i=1}^{n} y_i \log(p(y_i = 1 \mid \mathbf{w}, \mathbf{x}_i)) + (1 - y_i) \log(p(y_i = 0 \mid \mathbf{w}, \mathbf{x}_i)) \tag{5}$$

Plug in our logistic function for the probability that y is 1:

$$\ell(Y, X, \mathbf{w}) = \sum_{i=1}^{n} y_i \ \log(g(\mathbf{w}\mathbf{x}_i)) + (1 - y_i) \ \log(1 - g(\mathbf{w}\mathbf{x}_i))) \tag{6}$$

Recall that the link function, g, is the logistic function. It has the nice property $1 - g(z) = g(-z)$.

$$\ell(Y, X, \mathbf{w}) = \sum_{i=1}^{n} y_i \ \log(g(\mathbf{w}\mathbf{x}_i)) + (1 - y_i) \ \log(g(-\mathbf{w}\mathbf{x}_i)) \tag{7}$$

We can now use the chain rule to take the gradient with respect to $\mathbf{w}$:

$$\nabla\ell(Y, X, \mathbf{w}) = \sum_{i=1}^{n} y_i \ \frac{1}{g(\mathbf{w}\mathbf{x}_i)} \nabla g(\mathbf{w}\mathbf{x}_i) + (1 - y_i) \ \frac{1}{g(-\mathbf{w}\mathbf{x}_i)} \nabla g(-\mathbf{w}\mathbf{x}_i) \tag{8}$$

Since $\frac{\partial}{\partial z} g(z) = g(z)(1 - g(z))$:

$$\nabla\ell(Y, X, \mathbf{w}) \ = \ \sum_{i=1}^{n} y_i \ \frac{1}{g(\mathbf{w}\mathbf{x}_i)} g(\mathbf{w}\mathbf{x}_i)(1 - g(\mathbf{w}\mathbf{x}_i)) \nabla \mathbf{w}\mathbf{x}_i \tag{9}$$

$$+(1 - y_i) \ \frac{1}{g(-\mathbf{w}\mathbf{x}_i)} g(-\mathbf{w}\mathbf{x}_i)(1 - g(-\mathbf{w}\mathbf{x}_i)) \nabla(-\mathbf{w}\mathbf{x}_i) \tag{10}$$

Simplify again using $1 - g(z) = g(-z)$ and cancel terms

$$\nabla\ell(Y, X, \mathbf{w}) = \sum_{i=1}^{n} y_i g(-\mathbf{w}\mathbf{x}_i) \nabla \mathbf{w}\mathbf{x}_i + (1 - y_i) g(\mathbf{w}\mathbf{x}_i) \nabla(-\mathbf{w}\mathbf{x}_i) \tag{11}$$

You can now get the partial derivatives (components of the gradient) out of this gradient function by:

$$\frac{\partial}{\partial \mathbf{w}_j}\ell(Y, X, \mathbf{w}) = \sum_{i=1}^{n} y_i g(-\mathbf{w}\mathbf{x}_i)\ \mathbf{x}_{ij} + (1 - y_i)g(\mathbf{w}\mathbf{x}_i)(-\mathbf{x}_{ij}) \tag{12}$$

Remember that because we are doing stochastic gradient descent, we are not calculating the full gradient, which is the sum over all the data. We are sampling *one training instance* $(y_i, \mathbf{x}_i)$ and evaluating the gradient there. So for each SGD step, the partial derivatives for each weight $\mathbf{w}_j$ at one point are:

$$\nabla\ell(y_i, \mathbf{w}, \mathbf{x}_i) = \frac{\partial}{\partial \mathbf{w}_j}\ell(y_i, \mathbf{w}, \mathbf{x}_i) = y_i g(-\mathbf{w}\mathbf{x}_i)\mathbf{x}_{ij} + (1 - y_i)g(\mathbf{w}\mathbf{x}_i)(-\mathbf{x}_{ij}) \tag{13}$$

i is the instance index
j is the feature index

It is this equation that you will use to calculate your updates.

1.4 AdaGrad: Adaptively Choosing the Learning Rate

In SGD, the update rule is $\mathbf{w}' = \mathbf{w} + \eta\nabla\ell(y_i, \mathbf{w}, \mathbf{x}_i)$. Choosing η intelligently is a non-trivial task, and there are many ways to choose it in the literature. In this assignment, we will use an adaptive gradient method called *AdaGrad*[2], which compute a new η for each iteration based on historical information, so that frequently occurring features in the gradients get smaller learning rates and infrequent features get higher ones.

AdaGrad provides a per-feature learning rate $\eta_{i,j}$ at each time step i for feature j as:

$$\eta_{i,j} = \frac{\eta_0}{\sqrt{I_j + \sum_{t=1}^{i} f_{t,j}^2}} \tag{14}$$

where $f_{t,j}$ is the partial gradient of the objective function at time t for feature j (i.e., $f_{t,j} = \frac{\partial}{\partial \mathbf{w}_j}\ell(y_t, \mathbf{w}, \mathbf{x}_t)$). $0 \leq I_j \leq 1$ is the initial value for the denominator of $\eta_{i,j}$. A larger I_j will prevent $\eta_{1,j}$ from being too large if $f_{i,j}$ is too small. $\eta_0 > 0$ is a constant scalar. You can tune η_0 and I_j for better performance in terms of convergence rate.

As a result, the update rule for AdaGrad is $\mathbf{w}'_j = \mathbf{w}_j + \eta_{i,j}\nabla\ell(y_i, \mathbf{w}, \mathbf{x}_i)$.

In this assignment, we always *set* $I_j = 1$ for all j. We also use η_0. By default it should be 0.01, but your code should allow the user to change it by setting a command line argument called **sgd_eta0**.

1.5 Offset Feature

None of the math above mentions an offset feature (bias feature), a $\mathbf{w}_0$, that corresponds to a $x_{i,0}$ that is always 1. It turns out that we don't need this if our data is centered. By centered we mean that $E[y] = 0$. While this may or may not be true, for this assignment you should assume that your data is centered. Do not include another feature that is always 1 (x_0) or weight ($\mathbf{w}_0$) for it.

[2] http://www.magicbroom.info/Papers/DuchiHaSi10.pdf

1.6 Convergence

In real SGD, you must decide when you have converged. Ideally, a maximized function has a gradient value of 0, but due to issues related to your step size, random noise, and machine precision, your gradient will likely never be exactly zero. Usually people check that the L_p norm of the gradient is less than some δ, for some p. For the sake of simplicity and consistent results, we will not do this in this assignment. Instead, your program should take a parameter **sgd_iterations** which is *exactly* how many iterations you should run (not an upper bound). An iteration is a single pass over every training example. The default of **sgd_iterations** should be 20.

1.7 Implementation Notes

1. Many descriptions of SGD call for shuffling your data before learning. This is a good idea to break any dependence in the order of your data. In order to achieve consistent results for everyone, we are requiring that you **do not shuffle your data**. When you are training, you will go through your data in the order it appeared in the data file. If you require more iterations than there are lines in the file, then once you hit the last example you should loop around to the first example.

2. Even though logistic regression predicts a probability that the label is 1, the output of your program should be binary. Round your solution based on whether the probability is greater than or equal to 0.5:

   $\hat{y}_{new} = 1$ if $p(y = 1|\mathbf{w}, \mathbf{x}) = g(\mathbf{wx}) = \frac{1}{1+e^{-\mathbf{wx}}} \geq 0.5$
   $\hat{y}_{new} = 0$ otherwise

3. Initialize the parameters $\mathbf{w}$ to 0.

1.8 How Your Code Will Be Called

To train a model we will call:

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

There are some additional parameters which your program must support during training:

```
-sgd_eta0 k         // sets the the constant scalar, default = 1.0
-sgd_iterations t  // sets the number of SGD iterations, default = 20
```

All of these parameters are *optional*. If they are not present, they should be set to their default values.

To make predictions using a model we will call:

```
java cs475.Classify -mode test -algorithm logistic_regression \
        -model_file speech.logistic_regression.model \
        -data speech.test \
        -predictions_file speech.test.predictions
```

Remember that your output should be 0/1 valued, not real valued.

You can add a command line parameter by adding the following code block to the `createCommandLineOptions` method of `Classify`.

```
registerOption("sgd_eta0", "double", true, "The constant scalar for learning rate in AdaGrad.");
registerOption("sgd_iterations", "int", true, "The number of SGD iterations.");
```

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

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

```
int sgd_iterations = 20;
if (CommandLineUtilities.hasArg("sgd_iterations"))
    sgd_iterations = CommandLineUtilities.getOptionValueAsInt("sgd_iterations");
double sgd_eta0 = 1.0;
if (CommandLineUtilities.hasArg("sgd_eta0"))
    sgd_eta0 = CommandLineUtilities.getOptionValueAsFloat("sgd_eta0");
```

2 Analytical (50 points)

1) Fisher Linear Discriminant and Logistic Regression Classifiers (15 points) Generative models and discriminative models are somehow connected given certain scenarios. Suppose that we have samples from two classes with equal prior. The first class of samples have their features independent generated from a multivariate normal distribution $N(\mu_1, \Sigma)$, and the second class of samples have their features independently generated from a multivariate normal distribution $N(\mu_2, \Sigma)$.

(a) Prove that the class label y conditioning on the feature vector X follows a logistic regression model.

(b) Prove that the classifier based on the logistic regression model obtained in (a) is equivalent the optimal Fisher linear discriminant classifier. The optimal Fisher linear discriminant classifier is obtained using the population means and covariance matrix; see section 4.1.4 in Bishop.

Hint: You only need to show that both classifiers use the same decision rule.

2) Linear Models (10 points) Besides the least square estimators, machine learning researchers are also interested in another type of estimators – maximum likelihood estimators. Consider a linear model $y = X\beta + \epsilon$, where $X \in \mathbb{R}^{n\times d}$ is the design matrix, $y \in \mathbb{R}^n$ is the response vector, and $\epsilon \in \mathbb{R}^n$ is the random noise with each entry independently sampled from $N(0, \sigma^2)$. Please derive the maximum likelihood estimator of β and σ.

3) Regularization and Overfitting. (5 points) Statisticians love linear models because these models are very simple and interpretable. Many variants of linear models has been proposed, and most of them are formulated as (penalized) least squares program. Here we have three least squares programs,

$$\hat{\beta}_0 = \operatorname*{argmin}_{\beta_0} \|y - X_1\beta_0\|_2^2, \tag{15}$$

$$(\hat{\beta}_1, \hat{\beta}_2) = \operatorname*{argmin}_{\beta_1,\beta_2} \|y - X_1\beta_1 - X_2\beta_2\|_2^2, \tag{16}$$

$$\hat{\beta}_3 = \operatorname*{argmin}_{\beta_3} \|y - X_1\beta_3\|_2^2 + \lambda\|\beta_3\|_2^2, \tag{17}$$

where $\lambda > 0$, $y \in \mathbb{R}^n$, $X_1 \in \mathbb{R}^{n \times d_1}$, and $X_2 \in \mathbb{R}^{n \times d_1}$. (17) is well known as the ridge regression. The square norm acts as a penalty function to reduce overfitting. Prove

$$\|y - X_1\hat{\beta}_3\|_2^2 \geq \|y - X_1\hat{\beta}_0\|_2^2 \geq \|y - X_1\hat{\beta}_1 - X_2\hat{\beta}_2\|_2^2. \tag{18}$$

4) Decision Tree (10 points) Let's investigate how accurately decisions trees can learn. We start by constructing a unit square ($[0; 1] \times [0; 1]$). We select n samples from the square, each with a binary label ($+1$ or -1), such that no two samples share either x or y coordinates. Unlike the programming above, each feature can be used multiple times in a decision tree. At each node we can only conduct a binary threshold split using one single feature.

(a) Prove that we can find a decision tree of depth at most $\log_2 n$, which perfectly labels all n samples.

(b) If the samples can share either x or y coordinates but not both, can we still learn a decision tree which perfectly labels all n samples? Why or why not?

5) Conjugate Prior (10 points) The conjugate priors are very popular in Bayesian data analysis. The formal description of the conjugate priors can be found in Chapter 2.4.2. of Bishop's PRML.

(a) Prove that the Gamma distribution with parameters α and β is a conjugate prior of the Poisson distribution with parameter λ.

(b) Prove that the Beta distribution with parameters α and β is a conjugate prior of the geometric distribution with parameter p.

Hint: The easiest way to do this is to separate out the "interesting" part of the density from the normalizing constants.

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. Use the provided tex file for writing 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.