COSC74/174: Machine Learning and Statistical Data Analysis
Homework 3. Due: 11:59PM on Wednesday, 6 Nov 2019
Instructor: Prof. Soroush Vosoughi

Logistic Regression & Maximum Likelihood Estimation

You are given a training dataset in CSV format (hw3_data.csv). The files each contain 10000 rows with 3 columns.

Column 1 & 2 are the features.

Column 3 is class (binary, either 1 or 0).

Your goal for this assignment is to implement logistic regression using Maximum Likelihood Estimation.

Recall that the sigmoid function is used in logistic regression. The sigmoid function is given as:

$$\boldsymbol{P(y|x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\cdots+\beta_m x_m)}}}$$

Where $X = \{x_1 \ldots x_m\}$ correspond to the features and $\beta = \{\beta_0 \ldots \beta_m\}$ are the weights, with β_0 being the intercept.

Part a) [10 points] Implement the sigmoid function. The function should take the weights and the features as input and return a single value. Your function should be general and work for weight and feature vectors of any size.

```
def sigmoid(weights, features):

    return value
```

Don't forget, if you have m features, you will have m+1 weights (one is the intercept β_0)

The likelihood function for a binary classifier (such as logistic regression) can be treated as Bernoulli distribution. In other words, the likelihood function for logistic regression can be written as:

$$Likelihood(\beta) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$$

The function p here corresponds to the sigmoid function p(y|x) shown above. The likelihood function is a function of the β , which is our weight vector. The parameter n here is the number of data points you have (so you multiply the probabilities for each data point). Plugging in the formula for p(x) and taking the log (to get the log-likelihood) gives us the following equation:

$$\sum_{i=1}^{n} -log(1 + e^{\beta_0 + x_i.\beta}) + \sum_{i=1}^{n} y_i(\beta_0 + x_i.\beta)$$

This equation is for a simple case where you have only one feature (hence only two βs).

Part b) [15 points] Implement the log-likelihood function. The function should take weights, features and the given class labels as input and return a single number, the log-likelihood. Your function should be general and work for weight and feature vectors of any size.

```
def log_likelihood(weights,features,actual_ys):

    return log_likelihood
```

One way to train a logistic regression function is to find weights (β) that maximizes the log-likelihood function for it, hence called maximum likelihood estimation (I showed an example of this for linear regression in class). We already have the log_likelihood function (see above), to maximize it, we need to take its derivative. The partial derivative of the log-likelihood function for each weight, β_j is given as:

$$\frac{\partial ll}{\partial \beta_j} = \sum_{i=1}^{n} (y_i - p(x_i))x_{ij}$$

Here you loop over all n data points. $p(x_i)$ corresponds to the prediction. In other words, $p(x_i)$ is the output of the sigmoid function (given the current weights). y_i corresponds to the actual class label and x_{ij} corresponds to the jth feature of data row i. For instance, in the partial for β_1 , j would be 1 and so x_{ij} would correspond to the first feature of data row i.

The above equation works for all βs, except for β_0, which is the intercept. There is no corresponding feature for the intercept (i.e, no x is multiplied by it), therefore, for β_0 the partial derivative is:

$$\frac{\partial ll}{\partial \beta_0} = \sum_{i=1}^{n} (y_i - p(x_i))$$

Part c) [75 points]

Implement a function that learns the weights from our data using gradient descent (use the gradient formulas above). Your function should take in num_iterations and learning_rate as a parameter (along with the training data). Use the code skeleton below:

```
def learn_weights(actual_ys, features, num_iterations, learning_rate):
    initialize weights
    for i in range(num_iterations):
        calculate gradients for all weights
        update weights
        (note, each weight can be updated like this: weight[j]+=learning_rate * gradient[j])
        if i%1000==0:
            print(log_likelihood(weights,features,actual_ys))
    return weights
```

Use the following hyperparameters:

num_iterations = 10000

learning_rate = 0.00005

initial weights should be set to 0

Save the log-likelihoods for each 1000 iterations and plot them (use plt.plot()). What is the highest log-likelihood you can achieve?

Note that since this is the LOG-likelihood, the values will be negative. They should get bigger (get closet to 0) as you learn the weights. Also note that this will be slow, so don't think you have anything wrong if it takes a while to finish.

BONUS) [10 points]

Use the weights learned in the last part to create a logistic regression classifier. The logistic regression classifier should take a threshold T to decide whether a class is 1 or 0 (if p>T 1 else 0). Hint: Use the sigmoid function that you implemented in part a).