CIS 419/519 Applied Machine Learning
Homework 3: Logistic Regression

Due: February 19, 2020 11:59pm

Instructions

Collaboration: Make certain that you understand the collaboration policy described on the course website.

You should feel free to talk to other members of the class in doing the homework. I am more concerned that you learn how to solve the problem than that you demonstrate that you solved it entirely on your own. You may *discuss* the homework to understand the problems and the mathematics behind the various learning algorithms, but you must **write down your solutions yourself**. In particular, **you are not allowed to share problem solutions or your code with any other students.** We take plagiarism and cheating very seriously, and will be using software to detect academic dishonesty, so please don't do it.

You are also prohibited from posting any part of your solution to the internet, even after the course is complete. Similarly, please don't post this PDF file or the homework skeleton code to the internet.

Formatting: Your programming solutions must be implemented in python, following the specified API. A few parts of the programming exercise asks you to create plots or describe results; these should be included in your PDF document and submitted separately. All written solutions must be typeset using LaTeX.

Homework Template and Files: All files needed for this assignment are available online at:

- `https://www.seas.upenn.edu/~cis519/spring2020/homework/hw3_skeleton.ipynb`
- `https://www.seas.upenn.edu/~cis519/spring2020/data/hw3-diabetes.csv`
- `https://www.seas.upenn.edu/~cis519/spring2020/data/hw3-retinopathy.csv`
- `https://www.seas.upenn.edu/~cis519/spring2020/data/hw3-wdbc.csv`

Citing Your Sources: Any sources of help that you consult while completing this assignment (other students, textbooks, websites, etc.) ***MUST*** be noted in the your PDF file. This includes anyone you briefly discussed the homework with. If you received help from the following sources, you do not need to cite it: course instructor, course teaching assistants, course lecture notes, course textbooks or other readings.

Submitting Your Solution: You will submit this assignment through Gradescope as two parts: the written answers, and your python code for the programming exercise.

CIS 519 ONLY Problems: Some parts of the assignment will be marked as "[CIS 519 ONLY]". Only students enrolled in CIS 519 are required to complete these problems. However, we do encourage students in CIS 419 to read through these problems, although you are not required to complete them.

All homeworks will receive a percentage grade, but CIS 519 homeworks will be graded out of a different total number of points than CIS 419 homeworks. Students in CIS 419 choosing to complete CIS 519 ONLY exercises will not receive any credit for answers to these questions (i.e., they will not count as extra credit nor will they compensate for points lost on other problems).

Philosophy Behind This Homework In lecture recently, we have focused on the fundamentals of machine learning with basic models defined by a cost function and fit via gradient descent. Before we return to more applied work, it's essential to understand how these learning algorithms can be implemented, and so we'll be focusing on this aspect in this assignment, covering several ML algorithms. Once finished, you'll know in detail how to implement cost functions, gradient descent with convergence tests, basis expansion, regularization, and the effect of varying parameter choices and learning configurations (e.g., different types of regularization) on the resulting models. Establishing these foundations now will allow us to have later homeworks focus much more on applications.

PART I: PROGRAMMING EXERCISES

1 Logistic Regression (30 points)

Now that we've implemented a basic regression model using gradient descent in the last homework, we will use a similar technique to implement the logistic regression classifier.

1.1 Implementation

Implement regularized logistic regression by completing the `LogisticRegression` class. Your class must implement the following API:

- `__init__(alpha, regLambda, regNorm, epsilon, maxNumIters, initTheta)`: the constructor, which takes in α, λ, type of regularization (1 or 2), ϵ, and *maxNumIters* as arguments. *initTheta* is an optional parameter to set the initial value of theta.
- `fit(X,y)`: train the classifier from labeled data (X, y). X and y are Pandas data frames.
- `predict(X)`: return a predicted class for each instance in the Pandas data frame X
- `predict_proba(X)`: return class probabilities for each instance in the Pandas data frame X
- `computeCost(theta, X, y, regLambda)`: computes the logistic regression objective function for the given values of θ, X, y, and λ ("lambda" is a keyword in python, so we must call the regularization parameter something different). X and y are numpy matrices.
- `computeGradient(theta, X, y, regLambda)`: computes the d-dimensional gradient of the logistic regression objective for the given values of θ, X, y, and $regLambda = \lambda$. X and y are numpy matrices.
- `sigmoid(z)`: returns the sigmoid function of z

Note that these methods have already been defined correctly for you in `LogisticRegression`; be very careful not to change the API.

Sigmoid Function You should begin by implementing the `sigmoid(z)` function. Recall that the logistic regression hypothesis $h()$ is defined as:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where the sigmoid function $\sigma()$ is defined as:

$$\sigma(z) = \frac{1}{1 + \exp^{-z}}$$

The sigmoid function has the property that $g(+\infty) \approx 1$ and $g(-\infty) \approx 0$. Test your function by calling `sigmoid(z)` on different test samples. **Be certain that your sigmoid function works with both vectors and matrices** — for either a vector or a matrix, you function should perform the sigmoid function on every element.

Cost Function and Gradient Implement the functions to compute the cost function and its gradient. Recall the cost function for logistic regression with L2 regularization is a scalar value given by

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} [y_i \log(h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) + (1 - y_i) \log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_i))] + \lambda \|\boldsymbol{\theta}_{[1:d]}\|_2^2 \ .$$

The gradient of the cost function is a d-dimensional vector, where the j^{th} element (for $j = 1...d$) is given by

$$\frac{\partial \mathcal{J}(\boldsymbol{\theta})}{\partial \theta_j} = \sum_{i=1}^{n} (h_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i) x_{ij} + \lambda \theta_j \ .$$

We must be careful not to regularize the θ_0 parameter (corresponding to $x_{i0} = 1$), and so

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_0} = \sum_{i=1}^{n} (h_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i) \ .$$

Implement this for L2 regularization, and then derive and implement it for L1 regularization.

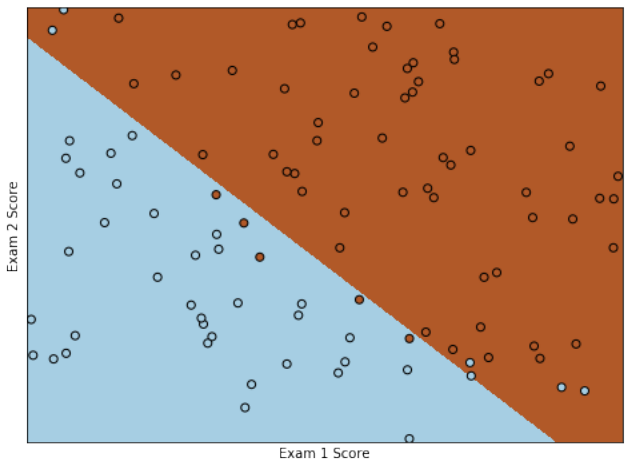
Training and Prediction Once you have the cost and gradient functions complete, implement the `fit` and `predict` methods. Make absolutely certain that the un-regularized θ_0 corresponds to the 1's feature that we add to the input.

Your `fit` method should train the model via gradient descent, relying on the cost and gradient functions. Instead of simply running gradient descent for a specific number of iterations (as in the linear regression code from the previous assignment), we will use the same sophisticated method we used within polynomial regression: we will stop it after the solution has converged. Stop the gradient descent procedure when $\boldsymbol{\theta}$ stops changing between consecutive iterations. You can detect this convergence when $\|\boldsymbol{\theta}_{new} - \boldsymbol{\theta}_{old}\|_2 \leq \epsilon$, for some small ϵ (e.g, ϵ = 1E-4). For readability, we'd recommend implementing this convergence test as a dedicated function `hasConverged`. For safety, we will also set the maximum possible number of gradient descent iterations, *maxNumIters*. The values of λ, ϵ, *maxNumIters*, and α (the gradient descent learning rate) are arguments to `LogisticRegression`'s constructor. At the start of gradient descent, $\boldsymbol{\theta}$ should be initialized to *initTheta* if specified or to random values with mean 0.

As you hopefully noticed, the logistic regression implementation is *very* similar to code from the previous homework assignment, so we can implement logistic regression reasonably quickly.

1.2 Testing Your Implementation

To test your logistic regression implementation, run `test_logreg1`. This script trains a logistic regression model using your implementation and then uses it to predict whether or not a student will be admitted to a school based on their scores on two exams. In the plot, the colors of the points indicate their true class label and the background color indicates the predicted class label. If your implementation is correct, the decision boundary should closely match the true class labels of the points, with only a few errors, as per below:



1.3 Analysis

Varying λ changes the amount of regularization, which acts as a penalty on the objective function for complex solutions. In `test_logreg1`, we provide the code to draw the decision boundary of the model. Vary the value of λ in `test_logreg1` and plot the decision boundary for each value. Also compare L1 vs L2 regularization. Include several plots in your PDF writeup for this assignment, labeling each plot with the value of λ and `regNorm` in your writeup. Write a brief paragraph (2-3 sentences) describing what you observe as you increase the value of λ and explain why, and what you observe with different regularization norms.

1.4 [CIS 519 ONLY] Learning a Nonlinear Decision Surface (10 pts)

We strongly encourage students in CIS 419 to read through this exercise in detail, even though you are not required to complete it. This exercise is required only for students in CIS 519.

Some classification problems that cannot be solved in a low-dimensional feature space can be separable in a high-dimension space. We can make our logistic regression classifier much more powerful by adding additional features to the input. Imagine that we are given a data set with only two features, x_1 and x_2, but the decision surface is non-linear in these two features. We can expand the possible feature set into higher dimensions by adding new features that are functions of the given features. For example, we could add a new feature that is the product of the original features, or any other mathematical function of those features.

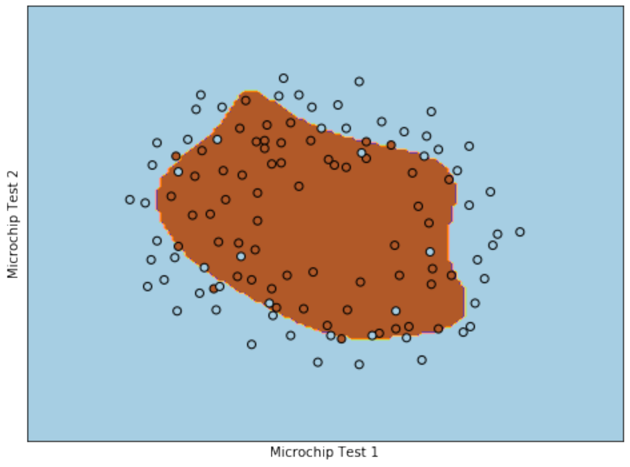
In this example, we will map the two features into all polynomial terms of x_1 and x_2 up to the 6th power:

$$mapFeature(X, column1, column2, 6) = \left(1, x_1, x_2, x_1^2, x_1 x_2, x_2^2, \cdots, x_1 x_2^5, x_2^6\right) \tag{1}$$

Given a 2-dimensional instance x, we can project that instance into a 28-dimensional feature space using the transformation above. Then, instead of training the classifier on instances in the 2-dimensional space, we train it on the 28-dimensional projection of those instances. The resulting decision boundary will then be more complex and will appear non-linear in the 2D plot.

Complete the `mapFeature` function to map instances from a 2D feature space to a feature space defined by all polynomial terms two different features up to the specified power. Your function `mapFeature(X, column1, column2, power)` will take in a Pandas data frame X, two column names, and the power for the expansion. Let $X1$ and $X2$ be the feature columns corresponding to those two column names. (Recall that X is n-by-d, and so both $X1$ and $X2$ will have n entries). It will return Pandas data frame, where each row represents the expanded feature set for one instance. It should work for at least $1 \leq power \leq 10$.

Once this function is complete, you can run `python test_logreg2`. This script will load a data set that is non-linearly separable, use your `mapFeature` function to project the instances into a higher dimensional space, and then train a logistic regression model in that higher dimensional feature space. When we project the resulting non-linear classifier back down into 2D, we get a decision surface that appears similar to:



2 Comparing Algorithms (30 pts)

In this section, you'll compare several algorithms with differing regularization methods to see how they perform. These methods include:

- Logistic Regression with L2 regularization – use your implementation from the previous exercise
- Logistic Regression with L1 regularization – use your implementation from the previous exercise
- Logistic Regression Adagrad – you will implement this

2.1 Logistic Regression Adagrad

Start by duplicating your implementation of logistic regression from the previous exercise into the class `LogisticRegressionAdagrad`. First, adjust your implementation to use stochastic gradient descent instead of regular gradient descent. The training should run multiple loops over the entire training set, selecting each instance in turn. The gradient is then computed for just that one instance, and the model parameters are updated accordingly based on that one gradient (not the average gradient).

Once you have stochastic gradient descent tested and working, recall the discussion of Adagrad from lecture. Adagrad adapts the learning rate to individual features, and changes it over time. Modify the gradient computation to use Adagrad instead of the constant learning rate alpha, allowing it to adapt the learning rate for each feature. Your implementation should work for both L1 and L2 regularization. Test your implementation thoroughly, convincing yourself that it works well.

2.2 Comparing Algorithms

There are numerous repositories of machine learning data sets on the internet, one of the most famous of which being the UCI Machine Learning repository (https://archive.ics.uci.edu/ml/index.php). Unfortunately, that means that these data sets are far overused for testing machine learning algorithms, and so we'll be avoiding them for any real applications. Nevertheless, they remain a good source of data to run simple comparisons between ML algorithms, as we'll do here.

Using three life sciences classification datasets from the UCI repository, we will compare the performance of several algorithms. Specifically, you should use the data sets listed below; download links for these datasets are in the homework instructions.

- Breast Cancer Wisconsin (Diagnostic): `hw3-wdbc.dat`. Please note that the patient ID has already been removed from the data set. The last column of the dataset indicates whether the lesion is benign (B) or malignant (M). Data originally from `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29`.
- Retinopathy Diagnosis: `hw3-retinopathy.dat`. The last column of the dataset indicates whether (1) or not (0) the patient exhibits signs of diabetic retinopathy. Data originally from `https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set`.
- Diabetes: `hw3-diabetes.dat` The last column indicates whether or not the patient tested positive for diabetes. Data from `https://www.kaggle.com/uciml/pima-indians-diabetes-database`.

For each data set, be sure to read the data set description and understand the features. Very briefly describe the dataset (number of features, number of instances, features and possible relationships among the features, does it have missing data, etc.).

On each data set, compare logistic regression with both L1 and L2 regularization, and logistic regression adagrad with both L1 and L2 regularization. If you wish, you may optionally include decision trees in your comparison, using the sklearn implementation. You should examine the performance of these methods over multiple trials of cross-validation, varying the parameters to understand thoroughly how they perform and why they perform that way. Report full summary statistics for the performance of these methods on the data sets. Be sure to state all aspects of your experimental setup, such that someone could exactly replicate your experiments.

2.3 [CIS 519 only] Understanding Regularization and Adagrad (10 pts)

In addition to computing summary performance, choose one data set and show the learning curves for the different forms of logistic regression. Your learning curves should depict the mean performance **during different steps within the gradient descent**. You can accomplish this by adjusting the maximum number of iterations to yield different models at different points along the learning curve. Note that we are NOT building learning curves for varying sizes of the training set; these learning curves will depict how the model performance changes over time as gradient descent runs. Be sure to vary the alpha for the non-adagrad logistic regression, since it will give you insight into how adagrad relates to choosing different alphas.

Include plots that compare multiple learning curves in your report, and write a few paragraphs analyzing how different choices of regularization, the learning rate, and the use of adagrad affect performance. Focus on the difference between adagrad and using a fixed learning rate.