This homework is from University of Washington Machine Learning Course cse446

Homework 2: Logistic Regression

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

- `__init__(alpha, regLambda, epsilon, maxNumIters)`: the constructor, which takes in α, λ, ϵ, and *maxNumIters* as arguments
- `fit(X,y)`: train the classifier from labeled data (X, y)
- `predict(X)`: return a vector of n predictions for each of n rows of 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)
- `computeGradient(theta, X, y, reg)`: computes the d-dimensional gradient of the logistic regression objective function for the given values of θ, X, y, and $reg = \lambda$
- `sigmoid(z)`: returns the sigmoid function of z

Note that these methods have already been defined correctly for you in `logreg.py`; 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:

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

where $g()$ is the sigmoid function defined as:

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

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.

This homework is from University of Washington Machine Learning Course cse446

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

$$\mathcal{J}(\boldsymbol{\theta}) = \sum_{i=1}^{n} [-y^{(i)} \log(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})) - (1 - y^{(i)}) \log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}))] + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 \ .$$

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

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

We must be careful not to regularize the θ_0 parameter (corresponding to the 1's feature we add to each instance), and so

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

Training and Prediction Once you have the cost and gradient functions complete, implement the `fit` and `predict` methods. To make absolutely certain that the un-regularized θ_0 corresponds to the 1's feature that we add to the input, we will augment both the training and testing instances **within** the `fit` and `predict` methods (instead of relying on it being done externally to the classifier). Recall that you can do this via:

```
X = np.c_[np.ones((n,1)), X]
```

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 exercise), we will use a more sophisticated method: 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

$$\|\boldsymbol{\theta}_{new} - \boldsymbol{\theta}_{old}\|_2 \leq \epsilon \ , \tag{6}$$

for some small ϵ (e.g, ϵ = 10E-4). For readability, we'd recommend implementing this convergence test as a dedicated function `hasConverged`. For safety, we will also set the maximum 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 random values with mean 0, as described in the linear regression exercise.

This homework is from University of Washington Machine Learning Course cse446

TESTING

To test your logistic regression implementation, run `python test_logreg1.py` from the command line. 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.