Middle East Technical University Department of Computer Engineering

CENG 499

Introduction to Machine Learning

Fall 2017-2018

Take Home Exam 1

Due date: **07.11.2016 - 23:59**

1 Objectives

To familiarize yourselves with linear classification, particularly the **logistic regression** through implementing and applying the classifier on a real data set, on which you may have to perform **feature scaling (normalization)**, and with analysis of error history.

2 Specifications

This homework invites you to perform logistic regression classification on the given data set, which will be described in detail subsequently.

Training data for this task is included in the file named "**pima-indians-diabetes.csv**" dated back to 1990. The dataset contains 768 data instances each comprising of 8 attributes followed by the class labels indicating the result of the diabetes test (1 for positive, and 0 for negative). Dataset is publicly available on UCI Machine Learning Repository, under the following link to which you may refer to regarding details.

`http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes`

In this assignment, your aim is to model a diabetes diagnostician using a logistic regressor whose parameters are learned via gradient descent algorithm using **the first 668 instances**. The remaining instances will be used for testing purposes, and you are **NOT** allowed to use test instances in training part.

Homework file continues with a review of theory behind logistic regression and the steps you should take into consideration for your implementation of the task, as organized in the two successive subsections.

2.1 Logistic Regression

Assume that you are given the training data set $T = \{(\mathbf{x}_i, y_i) \,|\, i \in [1..m]\}$, in which each $\mathbf{x}_i$ is an n-dimensional multivariate vector, and each $y_i \in \{0, 1\}$ denotes the class label, where 1 indicates that corresponding $\mathbf{x}_i$ is a positive instance, i.e. member of the class in question, while the latter implies that the data instance belongs to the negative class, namely it is not a member of the class in which we are interested.

Logistic regression is a linear classifier that incurs the sigmoid hypothesis function which is formulated as follows given such a two-class classification problem.

$$h_{\mathbf{w}}(\mathbf{x}) = p(y = 1|\mathbf{x}; \mathbf{w}) = g(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp[-\mathbf{w}^T\mathbf{x}]}, \tag{1}$$

in which $\mathbf{w} = [w^{(0)}, w^{(1)}, ...w^{(n)}]^T$, an $(n+1)$-dimensional parameter vector of the classifier including the bias (intercept) term, and y is the class label affirming that the instance belongs to the class in question implied by the setting $y = 1$ and $\mathbf{x} = [x^{(0)}, x^{(1)}, ..., x^{(n)}]^T$, the $(n+1)$-dimensional augmented vector such that $x^{(0)} = 1$ for each data vector.

Decision making using the specified hypothesis function is based on some simple thresholding procedure as in the following rule.

$$\hat{y} = \begin{cases} 1 \;\; iff \;\; h_{\mathbf{w}}(\mathbf{x}) \geq 0.5 \\ 0 \;\; otherwise \end{cases}, \tag{2}$$

in which $\hat{y}$ is the estimated class label for data instance $\mathbf{x}$ concerning a two-class classification problem.

Training of the logistic regression deals with the question of establishing the components of the parameter vector $\mathbf{w}$ in such a way that the "best" fit to the data is attained. Here degree-of-fitness is measured by means of utilizing cost functions. In logistic regression setting, we are going to use the following function for this purpose.

$$J(\mathbf{w}) = -\frac{1}{m}\left[\sum_{i=1}^{m} y_i \log\left(h_{\mathbf{w}}(\mathbf{x}_i)\right) + (1 - y_i)\log\left(1 - h_{\mathbf{w}}(\mathbf{x}_i)\right)\right]. \tag{3}$$

Having defined the objective function as in (3), training of a logistic regression classifier reduces to solving the subsequent optimization problem.

$$\mathbf{w}_{opt} = \underset{\mathbf{w}}{\operatorname{argmin}}\, J(\mathbf{w}). \tag{4}$$

Unconstrained optimization problem formulated in (4) can be solved using traditional gradient-descent approach whose iterative step is summarized in the following update rule.

$$\mathbf{w} := \mathbf{w} - \alpha \frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}). \tag{5}$$

Note that update rule in (5) corrects each component of the parameter vector simultaneously at an acceleration dictated by the hyper parameter α whose direction is opposite to that of the partial derivative until some convergence criterion is fulfilled. Partial derivative term included in this equation can be evaluated as follows.

$$\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i)\mathbf{x_i}. \tag{6}$$

Partial derivative whose analytical form is specified in (6) is computed as the average difference between the estimated class label and true nature of data instances weighted by values of corresponding features. Note that we have to compute this in parallel for all features, suggesting need for vectorized implementation as asked explicitly in the next subsection. For the derivation of the analytical form refer to the lecture notes.

2.2 Programming and Interpretation Tasks

First, you need to input the given file by proper Python functions or by your own reader into your own choice of variables to initiate the computation. Use of data design matrix $\mathbf{X}$ together with a vector of class labels is highly recommended.

Feature scaling is critical in machine learning and other optimization problems to eliminate the influence of attribute scale on results. Hence, perform **feature scaling** so that all attributes reside in similar intervals. You may use a variety of techniques regarding that. State in your report the technique you have adopted. A simple method can be Min-Max Scaling which is subtracting the minimum and dividing with the difference between the maximum and the minimum so that the values are between 0 and 1. Other examples may be mean normalization, standardization and scaling to unit length.

You should write a Python function that can compute the sigmoid of a matrix according to equation (1), consequently implementation involves very basic vectorization. Whenever parameter vector has already been estimated (set) by the classifier, you should be able to give your augmented data design matrix $\mathbf{X} = [\mathbf{1}, \mathbf{X}]$ weighted by the estimated parameters as an input to your sigmoid function and get the response out of the classifier in question in only one pass.

Next, you should implement Python **functions** that compute and output objective function value, gradient vector and accuracy using the given inputs in accordance with equations (3) and (6). All of your functions must be constructed in a **vectorized** fashion! For the accuracy formula refer to equation (7).

$$accuracy = (1 - loss_{0-1}/N) * 100, \tag{7}$$

where N is the number of test samples and $loss_{0-1}$ is the number of wrong predictions known as zero-one loss.

Now, you should train the classifier to estimate corresponding parameter vector, $\mathbf{w}$. For this purpose, implement the generic gradient-descent algorithm update rule of which is given in (5). For the hyper parameter α choose three different values; a small one, a moderate one and a large one, and repeat the training procedure for the three. An example set of values is [**3e-4, 1e-3, 1e-1**]. You are not restricted to use these values. Any set similar to this will be accepted as long as it includes a small, a moderate and a large value.

In order to start gradient descent algorithm we need an initial parameter vector $\mathbf{w}$ which is chosen to be comprised of small random numbers in general. You will follow this manner with a small difference which will make your results reproducible and comparable. Before starting each experiment on the α values set the random number generator's seed to "499" and get a random parameter vector. You can use *numpy.random.seed(499)* for NumPy or *random.seed(499)* for Python's random module.

Before training, divide your data into training and validation parts meaningfully and describe your justification for this division in your report. Note that you will use the first 668 instance for training and validation sets. The remaining ones will be used for testing. While training the classifier, keep histories of objective function value and prediction accuracy. Draw plots showing the change in objective function (the classification error) and the prediction accuracy over time (approximately 10000 updates) for training set and validation set. Discuss overfitting and underfitting issues, convergence time and local/global minimum issues using those plots. Note that, in order to see some meaningful information on the plots, you should do considerable amount of updates. Then, discuss the time point where training should be stopped.

The following bullet points summarize the key points that you should address in your report.

- The effect of the bias term that we add to form augmented data
- The feature scaling technique that you utilized, and the effects of it
- Discussion on the effects of the different learning rates
- Suggestion for a better learning rate selection method
- Detailed explanation of the decision about when to stop updates
- Detailed explanation of how you choose the training and validation sets
- Interpretations of the plots including the overfitting and underfitting issues
- Any other detail that needs to be explained

3 Restrictions and Tips

- Do not use any available Python repository files without referring to them in your report.
- Toolbox and library function use is not allowed in this homework. Do not use any ML-related toolbox/library. However, you may cross-check your results utilizing toolboxes. Your homework submission must not include any high-level toolbox/library function.
- If you encounter trouble regarding vectorization, first try to implement the tasks by using loops. Vectorization is required, since typical ML projects deal with massive amounts of data. If at the end you fail to vectorize your code, submit the current version. Although vectorization appears to be essential in Python-ML programming, since this is not a Python course, unvectorized versions will only result in a minor decline (at most 10%) in the grade you are going to receive.

- Implementation should be of your own. Readily-used codes should not exceed a reasonable threshold within your total work. In fact you shouldn't need any code on repositories.
- Don't forget that the code you are going to submit will also be subject to manual inspection.

4 Submission

- **Late Submission:** You have a total of 3 days for all the homeworks with a penalty of 10 points for each day.
- Your implementation files together with a 3-to-4 pages long report focusing on theoretical and practical aspects you observed regarding this task should be uploaded on COW before the specified deadline as a compressed archive file whose name should be **<student_id>_the1.tar.gz**, e.g. 1234567_the1.tar.gz.
- The archive must contain **no directories** on top of implementation files.
- Do not send the dataset in the archive. Evaluation will be done using the original dataset.

5 Regulations

1. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.
2. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.