Middle East Technical University | Department of Computer Engineering

CENG 499

Introduction to Machine Learning

Fall 2015-2016

Homework 2

Due date: Dec 25, 2015, Fri, 23:55

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** and **feature mapping** by using higher-order polynomials so that **nonlinear** classification may be utilized within a linear classification scheme, perform **cross-validation** on the provided data set and deduce whether some **overfitting** or **underfitting** phenomenon has occurred.

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 "**winequality-white.csv**" dated back to 2009, whose first row carries information on the name of attributes, in which the last entity constitutes the class name in question. Following lines contain the 4898 data instances each comprising of 11 attributes followed by the class labels indicating the quality of wine as an integer. Data set is publicly available on UCI Machine Learning Repository, under this link to which you may refer to regarding details.

In this assignment, your aim will be modelling the quality of white wine as a function of given attributes which are essentially results of conducted physicochemical tests.

Homework file is continued with a review of theory behind logistic regression and the steps you should take concerning 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, yet may be a member of the other class(es).

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

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

in which $\theta = [\theta^{(0)}, \theta^{(1)}, ...\theta^{(n)}]^T$, 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 $x = [x^{(0)}, x^{(1)}, ..., x^{(n)}]^T$, 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.

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

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 θ in such a way that the "best" fit to the data is attined. 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.

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

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

$$\theta_{opt} = \underset{\theta}{\operatorname{argmin}} \, J(\theta) \tag{4}$$

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

$$\theta_i := \theta_i - \alpha \frac{\partial}{\partial_i} J(\theta)\,, \;\; \forall i \tag{5}$$

Note that update rule in (5) corrects each component of the parameter vector simultaneously at an acceleration dictated by the 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.

$$\frac{\partial}{\partial_i} J(\theta) = \frac{1}{m} \sum_{j=1}^{m} (h_\theta(\mathbf{x}_j) - y_j) x_j^{(i)} \tag{6}$$

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.

2.2 Programming Tasks

Change the default format of the MATLAB workspace into long fixed-decimal format to avoid possible numerical errors.

First, you need to input the given file by proper MATLAB functions such as **load** command into your own choice of variables to initiate the computation. Use of data design matrix $\mathbf{X}$ together with a vector of class labels as described in the previous assignment is highly recommended.

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. You may also manually **select** a subset of features that are tested provide better prediction accuracy. State which features you are going to discard and the reason behind your action.

You should write a MATLAB **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 another MATLAB **function** that takes data design matrix, vector of labels, and parameters vector as inputs and computes and outputs objective function value and gradient vector comprised of partial derivatives of parameter vector using the given inputs in accordance with equations (3) and (6). Both of your functions must be constructed in a **vectorized** fashion!

Note that you have more classes than just two in this problem. Thus, you have to train C logistic regression classifiers, where C is the number of classes. For this purpose you may generate C copies of your class label vector, each indexed as $\mathbf{y}_k$, so that labels are rewritten as 1's for instances with label k and $0's$ otherwise in accordance with the two-class classification problem formulation of the previous section.

Now, you should train each classifier to estimate corresponding parameter vectors, θ_k. For this purpose, instead of implementing the generic gradient-descent algorithm, you are going to use an optimized MATLAB function, namely `fminunc` which conducts a more intelligent optimization procedure called "line-search" within its inner loop and converges faster than the gradient-descent. Use of this command will be more clear in the following, assuming that you have implemented the objective function as described previously.

```
thetaOpt = fminunc(@yourCostFunction, someInitialTheta, 'GradObj', 'on');
```

Here, you will provide the built-in MATLAB optimization function with inputs as a handle to your objective function that returns both value of the function and gradient vector, some initial vector to initialize the procedure, together with setting an option that indicates whether the analytical form of the gradient is present, respectively. There are other options, including setting the limit of maximum iterations until convergence. You should check the MATLAB documentation before actually using the function.

In this manner, you will end up finding optimum values of θ_k for each classifer. This ends the training procedure for the given data set.

Predicting the class an instance belongs to needs to be modified for multi-class classification. This procedure will be based on selecting the class whose corresponding classifier's hypothesis response is **maximal** among all you have designed.

Now, you have to evaluate the performance of the classifier having implemented the updated prediction procedure. Perform P-fold **cross-validation** utilizing the data available to you where P should be a "reasonable" integer, greater than 5. Calculate the average prediction error. Comment on the results you obtained in your report. State whether underfitting or overfitting has occurred by first calculating the error on the whole training set.

It is time to increase the number of features and introduce nonlinearity to our system. Conduct a second-degree polynomial expansion on the features selected in the first part. Through this **feature mapping** you will end up with $O(n^2)$ features as compared to the original n features. Follow the same training and evaluation steps and obtain another average prediction error. How did the classifier accuracy change? State whether underfitting or overfitting has occurred again by computing the error on the entire data set. What could be the remedy to underfitting or overfitting? Write your analysis of the task in detail in the report you are going to submit.

3 Restrictions and Tips

- Do not use any available MATLAB repository files without referring to them in your report.
- Toolbox function use is not allowed in this homework. Do not use any ML-related toolbox. However, you may cross-check your results utilizing toolboxes. Your homework submission must not include any high-level toolbox 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 MATLAB programming, since this is not a MATLAB course, unvectorized versions will only result in a minor decline (at most 10%) in the grade you are going to receive.
- Use of normal equations is **forbidden**!
- 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:** No late submission is accepted.
- Your scripts and function 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**>_**hw2**> preceding the file extension.
- The archive must contain **no directories** on top of MATLAB/Octave scripts and function files.

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.