CSE474/574 Introduction to Machine Learning
Programming Assignment 3
Handwritten Digits Classification (continue)

Due Date: 11:59PM May 7^{th} 2014

1 Introduction

In this assignment, we will extend the first programming assignment in solving the problem of handwritten digit classification. In particular, your task is to implement Logistic Regression and use Support Vector Machine toolbox to classify hand-written digit images and compare the performance of these methods.
To get started with the exercise, you will need to download the supporting files and unzip its contents to the directory you want to complete this assignment.

1.1 Datasets

In this assignment, we still use the same data set of first programming assignment - MNIST. We have provided a file you *preprocess.m* which already implemented preprocessing steps (apply feature selection and feature normalization) and divide data set into 3 parts: training set, validation set and testing set.

2 Your tasks

- Implement **Logistic Regression** and give the prediction results.
- Use **Support Vector Machine (SVM)** toolbox to perform classification.
- Write a report to explain the experimental results with these 2 methods.

2.1 Logistic Regression

2.1.1 Formula derivation

Suppose that a given vector $\mathbf{x} \in \mathbb{R}^D$ is any input vector and we want to classify x into correct class C_1 or C_2. In Logistic Regression, the posterior probability of class C_1 can be written as follow:

$$y = P(C_1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0)$$

where $\mathbf{w} \in \mathbb{R}^D$ is the weight vector.
For simplicity, we will denote $\mathbf{x} = [1, x_1, x_2, \cdots, x_D]$ and $\mathbf{w} = [w_0, w_1, w_2, \cdots, w_D]$. With this new notation, the posterior probability of class C_1 can be rewritten as follow:

$$P(C_1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) \tag{1}$$

And posterior probability of class C_2 is:

$$P(C_2|\mathbf{x}) = 1 - P(C_1|\mathbf{x})$$

We now consider the data set $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ and corresponding label $\{t_1, t_2, \cdots, t_N\}$ where

$$t_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \in C_1 \\ 0 & \text{if } \mathbf{x}_i \in C_2 \end{cases}$$

for $i = 1, 2, \cdots, N$.
With this data set, the likelihood function can be written as follow:

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n}$$

where $y_n = \sigma(\mathbf{w}^t \mathbf{x}_n)$ for $n = 1, 2, \cdots, N$.
We also define the error function by taking the negative logarithm of the log likelihood, we gives the cross-entropy error function of the form:

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \tag{2}$$

The gradient of error function with respect to w can be obtained as follow:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\mathbf{x}_n \tag{3}$$

Up to this point, we can use again - gradient descent - to find the optimal weight $\hat{\mathbf{w}}$ to minimize the error function with the formula:

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \gamma \nabla E(\mathbf{w}^{old}) \tag{4}$$

2.1.2 Implementation

You are asked to implement Logistic Regression to classify hand-written digit images into correct corresponding labels. In particular, you have to build 10 binary-classifiers (one for each class) to classify that class from all other classes. In order to implement Logistic Regression, you have to complete function *blrObjFunction.m* provided in the base code. The input of *blrObjFunction.m* includes 3 parameters:

- $\mathbf{X}$ is a data matrix where each row contains a feature vector in original coordinate (not including the bias 1 at the beginning of vector). In other words, $\mathbf{X} \in \mathbb{R}^{N \times D}$. So you have to add the bias into each feature vector inside this function. In order to guarantee the consistency in the code and utilize automatic grading, please add the bias at the beginning of feature vector instead of the end.
- $\mathbf{w}$ is a column vector representing the parameters of Logistic Regression. Size of $\mathbf{w}$ is $(D + 1) \times 1$.
- $\mathbf{t}$ is a column vector representing the labels of corresponding feature vectors in data matrix $\mathbf{X}$. Each entry in this vector is either 1 or 0 to represent whether the feature vector belongs to a class C_k or not $(k = 1, 2, \cdots, 10)$. Size of $\mathbf{t}$ is $N \times 1$ where N is the number of rows of $\mathbf{X}$.

Function *blrObjFunction.m* has 2 outputs:

- *error* is a scalar value which is the result of computing equation (2)
- **error_grad** is a column vector of size $(D + 1) \times 1$ which represents the gradient of error function obtained by using equation (3).

For prediction using Logistic Regression, given 10 weights vector of 10 classes, we need to classify a feature vector into a certain class. In order to do so, given a feature vector $\mathbf{x}$, we need to compute the posterior probability $P(C_k|\mathbf{x})$ and the decision rule is to assign $\mathbf{x}$ to class C_k that maximizes $P(C_k|\mathbf{x})$. In particular, you have to compltete the function *blrPredict.m* which returns the predicted label for each feature vector. Concretely, the input of *blrPredict.m* includes 2 parameters:

- Similar to function *blrObjFunction.m*, $\mathbf{X}$ is also a data matrix where each row contains a feature vector in original coordinate (not including the bias 1 at the beginning of vector). In other words, $\mathbf{X}$ has size $N \times D$. In order to guarantee the consistency in the code and utilize automatic grading, please add the bias at the beginning of feature vector instead of the end.
- $\mathbf{W}$ is a matrix where each column is a weight vector of classifier k. Concretely, $\mathbf{W}$ has size $(D+1) \times K$ where $K = 10$ is the number of classifiers.

The output of function *blrPredict.m* is a column vector **label** which has size $N \times 1$, in which a feature vector is classify to digit k will have label $k + 1$.

2.1.3 Extra credits

2.1.3.1 Logistic Regression with Hessian matrix

With error function given in equation (2), instead of using gradient descent to find the optimal weight to minimize the error function, we can use another iterative method - Newton Raphson method - to find the minimum of error function. The general update rule of this method can be written as follow:

$$\mathbf{w}^{new} = \mathbf{w}^{old} - H^{-1}\nabla E(\mathbf{w}^{old}) \tag{5}$$

where H is the Hessian matrix.
The details of this iterative method is explained in Bishop's textbook [2, pp 207-208]. Please complete function *blrNewtonRaphsonLearn.m* given in the base code. Notice that in this function, you have to implement equation (5) by yourself instead of using *fmincg.m*.

2.1.3.2 Multiclass Logistic Regression with Gradient Descent

In this part, you are asked to implement multiclass logistic regression. Traditionally, Logistic Regression is used for binary classification. However, Logistic Regression can also be extended to solve the multiclass classification. With this method, we don't need to build 10 classifiers like before. Instead, we now only need to build 1 classifier that can classify 10 classes at the same time. The explanation of multiclass logistic regression is given in Bishop's textbook [2, pp 209-210]. Concretely, you need to complete *mlrObjFunction.m* and *mlrPredict.m* using equation (4.108) and (4.109) in Bishop 's textbook.

2.1.3.3 Multiclass Logistic Regression with Hessian matrix

Similar to Logistic Regression for binary classification, instead of using gradient descent to find the optimal weight of error function for multiclass logistic regression, we can also use Newton-Raphson method to obtain the optimal weight. The details of this approach can be found in Bishop's textbook [2, pp 209-210]. In particular, you need to complete function *mlrNewtonRaphsonLearn.m*. Please notice that you cannot use *fmincg.m* to run the optimization part for this section any more. So please implement the equation (5) by yourself.

2.2 Support Vector Machine

In this part of assignment, you are asked to use *LibSVM* toolbox [3] to perform classification on our data set. We have already provided for you the *LibSVM* compiled files in folder *libsvm*. In order to use *LibSVM*, you have to copy all files in folder *libsvm/[your_platform]/* into the folder *base code*.
The command in Matlab to train an SVM model is:

```
model = svmtrain(train_label, train_data, [libsvm_options]);
```

In the above command, some relevant options of *libsvm_options* that you need to use to perform experiments are[1]:

[1] we only list here the options which are relevant to our project, for complete enumeration of options in SVM, please read README file from LibSVM

- -t kernel_type : set type of kernel function (default 2)
  - ✓ 0 – linear: $u^T v$
  - ✓ 2 – radial basis function: $exp(-gamma * |u - v|^2)$
- -c cost : set the parameter C of C-SVC (default 1)
- -g gamma : set gamma in kernel function (default 1/num_features)

For example, we can train a SVM model using linear kernel with parameter C setting to 50 using the following command:

```
model = svmtrain(train_label, train_data, `-t 0 -c 50');
```

After having the SVM model obtained by using svmtrain, we will need to use this model to perform classification by using the following command:

```
[predicted_label, accuracy, ~] = svmpredict(label, data, model);
```

Concretely, your task is to fill the code in Support Vector Machine section of *script.m* to learn the SVM model and compute accuracy of prediction with respect to training data, validation data and testing using the following parameters:

- Using linear kernel (all other parameters are kept default).
- Using radial basis function with value of gamma setting to 1 (all other parameters are kept default).
- Using radial basis function with value of gamma setting to default (all other parameters are kept default).
- Using radial basis function with value of gamma setting to default and varying value of C $(1, 10, 20, 30, \cdots, 100)$ and plot the graph of accuracy with respect to values of C in the report.

3 Submission

You are required to submit an only file *proj3.zip* to CSE server by using the following script:
submit_cse474 proj1.zip (for undergraduate students)
submit_cse574 proj1.zip (for graduate student)

File *proj3.zip* must contain 2 folders: *report* and *code*.

- Folder *report* contains your report file (in pdf format).
- Folder *code* must contains the following files:
  - (Required) *blrObjFunction.m*
  - (Required) *blrPredict.m*
  - (Required) *script.m*: your code in SVM section
  - (Required) *params.mat*: this file must contain the following variable: *W_blr* (weight matrix of size $(D + 1) \times 10$ where each column is a weight vector of a classifier in Logistic Regression), *model_linear* (SVM model obtained with linear kernel), *model_rbf_1* (SVM model obtained with radial basis function and setting gamma to 1), *model_rbf_default* (SVM model obtained with radial basis function and default value of gamma), *model_rbf_C* (best SVM model obtained when using radial basis function with default value of gamma and varying value of C).
  - (Optional) *blrNewtonRaphsonLearn.m*

- (Optional) *mlrObjFunction.m*
- (Optional) *mlrPredict.m*
- (Optional) *mlrNewtonRaphsonLearn.m*

Project report: The hard-copy of report will be collected in class at due date. Your report should include the experimental results you have performed using Logistic Regression and Support Vector Machine.

4 Grading scheme

- Implementation:
  - ✓ *blrObjFunction.m*: 20 points
  - ✓ *blrPredict.m*: 20 points
  - ✓ *script.m*: 20 points (your code in SVM section)
  - ✓ *blrNewtonRaphsonLearn.m*: 10 points (extra credits)
  - ✓ *mlrObjFunction.m* + *mlrPredict.m*: 20 points (extra credits)
  - ✓ *mlrNewtonRaphsonLearn.m*: 20 points (extra credits)
- Project report: 30 points
- Accuracy of classification methods: 10 points

References

[1] LeCun, Yann; Corinna Cortes, Christopher J.C. Burges. "MNIST handwritten digit database".

[2] Bishop, Christopher M. "Pattern recognition and machine learning (information science and statistics)." (2007).

[3] Chang, Chih-Chung, and Chih-Jen Lin. "LIBSVM: a library for support vector machines." ACM Transactions on Intelligent Systems and Technology (TIST) 2.3 (2011): 27, `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.