CSE574 Introduction to Machine Learning
Programming Assignment 4
Classification and Regression

Due Date: December 9^{th}2024
Maximum Score: 100 + 20

Note

A zipped file containing skeleton Python script files and data is provided. Note that for each problem, you need to write code in the specified function within the Python script file. **For logistic regression, do not use any Python libraries/toolboxes, built-in functions, or external tools/libraries that directly perform the learning or prediction.** Using any external code will result in 0 points for that problem.

Evaluation

We will evaluate your code by executing `script.py` file, which will internally call the problem specific functions. You must submit an assignment report (pdf file) summarizing your findings. In the problem statements below, the portions under REPORT heading need to be discussed in the assignment report.

Data Sets

In this assignment, we still use MNIST. In the script file provided to you, we have implemented a function, called *preprocess()*, with preprocessing steps. This will apply feature selection, feature normalization, and divide the dataset into 3 parts: training set, validation set, and testing set.

Submission

You are required to submit a single file called ***proj3.zip*** using UBLearns/Brighspace. File ***proj3.zip*** must contain 2 files: ***report.pdf*** and ***script.py***

- Submit your report in a pdf format. Please indicate the **team members** on the top of the report.
- The code file should contain all implemented functions. Please do not change the name of the file.

Using UBLearns/Brightspace Submission: You should submit one solution per group through the UBLearns/Brightspace page.

1 Your tasks

- Implement **Logistic Regression** and give the prediction results.

- Use the **Support Vector Machine (SVM)** toolbox `sklearn.svm.SVM` to perform classification.
- Write a report to explain the experimental results with these 2 methods.
- *Extra credit*: Implement the gradient descent minimization of multi-class **Logistic Regression** (using softmax function).

1.1 Problem 1: Implementation of Logistic Regression (40 code + 15 report = 55 points)

You are asked to implement Logistic Regression to classify hand-written digit images into correct corresponding labels. The data is the same that was used for the second programming assignment. Since the labels associated with each digit can take one out of 10 possible values (multiple classes), we cannot directly use a binary logistic regression classifier. Instead, we employ the *one-vs-all* strategy. In particular, you have to build 10 binary-classifiers (one for each class) to distinguish a given class from all other classes.

1.1.1 Implement *blrObjFunction()* function (20 points)

In order to implement Logistic Regression, you have to complete function *blrObjFunction()* provided in the base code (*script.py*). 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**.
- $\mathbf{w}_k$ is a column vector representing the parameters of Logistic Regression. Size of $\mathbf{w}_k$ is $(D+1) \times 1$.
- $\mathbf{y}_k$ 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 $(k = 0, 1, \cdots, K-1)$. Size of $\mathbf{y}_k$ is $N \times 1$ where N is the number of rows of $\mathbf{X}$. The creation of $\mathbf{y}_k$ is already done in the base code.

Function *blrObjFunction()* 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).

1.1.2 Implement *blrPredict()* function (20 points)

For prediction using Logistic Regression, given 10 weight vectors 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 $P(y = C_k|\mathbf{x})$ and the decision rule is to assign $\mathbf{x}$ to class C_k that maximizes $P(y = C_k|\mathbf{x})$. In particular, you have to complete the function *blrPredict()* which returns the predicted label for each feature vector. Concretely, the input of *blrPredict()* includes 2 parameters:

- Similar to function *blrObjFunction()*, $\mathbf{X}$ 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 $(\mathbf{w}_k)$ of classifier for digit k. Concretely, $\mathbf{W}$ has size $(D+1) \times K$ where $K = 10$ is the number of classifiers.

The output of function *blrPredict()* is a column vector **label** which has size $N \times 1$.

1.1.3 Report (15 points)

In your report, you should train the logistic regressor using the given data **X** (Preprocessed feature vectors of MNIST data) with labels **y**. Record the total error with respect to each category in both training data and test data. And discuss the results in your report and explain why there is a difference between training error and test error.

1.2 For Extra Credit: Multi-class Logistic Regression (10 code + 10 report = 20 points)

In this part, you are asked to implement multi-class Logistic Regression. Traditionally, Logistic Regression is used for binary classification. However, Logistic Regression can also be extended to solve the multi-class 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.

1.2.1 Implement *mlrObjFunction()* function (10 points)

In order to implement Multi-class Logistic Regression, you have to complete function *mlrObjFunction()* provided in the base code (*script.py*). The input of *mlrObjFunction.m* includes the same definition of parameter as above. Function *mlrObjFunction()* has 2 outputs that has the same definition as above. You should use multi-class logistic function to regress the probability of each class.

1.2.2 Report (10 points)

In your report, you should train the logistic regressor using the given data **X**(Preprocessed feature vectors of MNIST data) with labels **y**. Record the total error with respect to each category in both training data and test data. And discuss the results in your report and explain why there is a difference between training error and test error. Compare the performance difference between multi-class strategy with *one-vs-all* strategy.

1.3 Support Vector Machines (20 code + 25 report = 45 points)

In this part of assignment you are asked to use the Support Vector Machine tool in `sklearn.svm.SVM` to perform classification on our data set. The details about the tool are provided here: `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`.

1.3.1 Implement *script.py* function (10 points)

Your task is to fill the code in Support Vector Machine section of *script.py* 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.

1.3.2 Report (25 points)

In your report, you should train the SVM using the given data **X**(Preprocessed feature vectors of MNIST data) with labels **y**. And discuss the performance differences between linear kernel and radial basis, different gamma setting.

Appendices

A Logistic Regression

Consider $\mathbf{x} \in \mathbb{R}^D$ as an input vector. We want to classify $\mathbf{x}$ into correct class C_1 or C_2 (denoted as a random variable y). In Logistic Regression, the posterior probability of class C_1 can be written as follow:

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

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:

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

And posterior probability of class C_2 is:

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

We now consider the data set $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ and corresponding label $\{y_1, y_2, \cdots, y_N\}$ where

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

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

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

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

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

Note that this function is different from the squared loss function that we have used for Neural Networks and Perceptrons.

The gradient of error function with respect to $\mathbf{w}$ can be obtained as follow:

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

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

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

B Multi-Class Logistic Regression

For multi-class Logistic Regression, the posterior probabilities are given by a softmax transformation of linear functions of the feature variables, so that

$$P(y = C_k|\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T\mathbf{x})}{\sum_j \exp(\mathbf{w}_j^T\mathbf{x})} \tag{5}$$

Now we write down the likelihood function. This is most easily done using the 1-of-K coding scheme in which the target vector $\mathbf{y}_n$ for a feature vector $\mathbf{x}_n$ belonging to class C_k is a binary vector with all elements zero except for element k, which equals one. The likelihood function is then given by

$$P(\mathbf{Y}|\mathbf{w}_1, \cdots, \mathbf{w}_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} P(y = C_k|\mathbf{x}_n)^{y_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} \theta_{nk}^{y_{nk}} \tag{6}$$

where θ_{nk} is given by (5) and Y is an $N \times K$ matrix (obtained using 1-of-K encoding) of target variables with elements y_{nk}. Taking the negative logarithm then gives

$$E(\mathbf{w}_1, \cdots, \mathbf{w}_K) = -\ln\ P(\mathbf{Y}|\mathbf{w}_1, \cdots, \mathbf{w}_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln\ \theta_{nk} \tag{7}$$

which is known as the cross-entropy error function for the multi-class classification problem.

We now take the gradient of the error function with respect to one of the parameter vectors $\mathbf{w}_k$. Making use of the result for the derivatives of the softmax function, we obtain:

$$\frac{\partial E(\mathbf{w}_1, \cdots, \mathbf{w}_K)}{\partial \mathbf{w}_k} = \sum_{n=1}^{N} (\theta_{nk} - y_{nk})\mathbf{x}_n \tag{8}$$

then we could use the following updating function to get the optimal parameter vector w iteratively:

$$\mathbf{w}_k^{new} \leftarrow \mathbf{w}_k^{old} - \eta \frac{\partial E(\mathbf{w}_1, \cdots, \mathbf{w}_K)}{\partial \mathbf{w}_k} \tag{9}$$