CSE474 Introduction to Machine Learning
Programming Assignment 1

Linear Models for Supervised Learning

Due Date: **March 6th 2019**
Maximum Score: 100

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 withing the Python script file. **Do not use any Python libraries/toolboxes, built-in functions, or external tools/libraries that directly perform classification, regression, function fitting, etc.**. Using any ML library calls, such as `scikit-learn`, will result in 0 points for the corresponding problem.

Evaluation We will evaluate your code by executing `PA1Script.ipynb` file, which will internally call the problem specific functions. Also 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 Two data sets are provided:

1. A medical data set is provided in the file "diabetes.pickle" along with the target assignment. The input variables correspond to measurements (physical, physiological, and blood related) for a given patient and the target variable corresponds to the level of diabetic condition in the patient. It contains:
   - $\mathbf{x_{train}}$ (242×64) and $\mathbf{y_{train}}$ (242×1) for training.
   - $\mathbf{x_{test}}$ (200×64) and $\mathbf{y_{test}}$ (200×1) for testing.
2. A 2D sample data set in the file "sample.pickle".

Submission You are required to submit a single file called ***proj1.zip*** using UBLearns.
File ***proj1.zip*** must contain 2 files: ***report.pdf*** and ***PA1Script.ipynb***.

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

Please make sure that your group is enrolled in the UBLearns system: You should submit one solution per group through the groups page. *If you want to change the group, contact the instructors.*

Project report: The hard-copy of report will be collected in class at due date. The problem descriptions specify what is needed in the report for each problem.

Part I - Linear Regression

In this part you will implement the direct and gradient descent based learning methods for Linear Regression and compare the results on the provided "diabetes" dataset.

Problem 1: Linear Regression with Direct Minimization (5 code + 5 report = 10 Points)

Implement *ordinary least squares* method to estimate regression parameters by minimizing the squared loss.

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \tag{1}$$

In matrix-vector notation, the loss function can be written as:

$$J(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) \tag{2}$$

where $\mathbf{X}$ is the input data matrix, $\mathbf{y}$ is the target vector, and $\mathbf{w}$ is the weight vector for regression.

You need to implement the function `learnOLERegression`. Also implement the function `testOLERegression` to apply the learnt weights for prediction on both training and testing data and to calculate the *root mean squared error* (RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2} \tag{3}$$

REPORT 1.

Calculate and report the RMSE for training and test data for two cases: first, without using an intercept (or bias) term, and second with using an intercept. Which one is better?

Problem 2: Using Gradient Descent for Linear Regression Learning (15 code + 5 report = 20 Points)

As discussed in class, regression parameters can be calculated directly using analytical expressions (as in Problem 1). However, to avoid computation of $(\mathbf{X}^\top\mathbf{X})^{-1}$, another option is to use gradient descent to minimize the loss function. In this problem, you have to implement the gradient descent procedure for estimating the weights $\mathbf{w}$, where the gradient is given by:

$$\nabla J(\mathbf{w}) = \mathbf{X}^\top\mathbf{X}\mathbf{w} - \mathbf{X}^\top\mathbf{y} \tag{4}$$

You need to use the `minimize` function (from the **scipy** library). You need to implement a function `regressionObjVal` to compute the regularized squared error (See (2)) and a function `regressionGradient` to compute its gradient with respect to $\mathbf{w}$. In the main script, this objective function and the gradient function will be used within the minimizer (See `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html` for more details).

REPORT 2.

Using `testOLERegression`, calculate and report the RMSE for training and test data after gradient descent based learning. Compare with the RMSE after direct minimization. Which one is better?

Part II - Linear Classifiers

In this part you will implement three different linear classifiers using different optimization algorithms and compare the results on the provided data set. You will also have to draw the discrimination boundaries for the three classifiers and compare.

The three classifiers are:

1. Perceptron
2. Logistic Regression
3. Linear Support Vector Machine (SVM)

For each classifier, the decision rule is the same, i.e., the target, y_i, for a given input, $\mathbf{x}_i$ is given by:

$$y_i = \begin{cases} -1 & \text{if } \mathbf{w}^\top \mathbf{x}_i < 0 \\ +1 & \text{if } \mathbf{w}^\top \mathbf{x}_i \geq 0 \end{cases} \tag{5}$$

where $\mathbf{w}$ are the weights representing to the linear discriminating boundary. We will assume that we have included a constant term in $\mathbf{x}_i$ and a corresponding weight in $\mathbf{w}$. While all three classifiers have the same decision function[1]

For this part, you will implement the training algorithms for the three different linear classifiers, learn a model for the sample training data and report the accuracy on the sample training and test data sets. The sample training and test data sets are included in the "sample.pickle" file.

Problem 3: Using Gradient Descent for Perceptron Learning (10 code + 5 report = 15 points)

For this problem, you will training a perceptron, which has a squared loss function which is exactly the same as linear regression (See (1)), i.e.,

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \tag{8}$$

which means that you can call the same functions, `regressionObjVal` and `regressionGradient`, implemented in Problem 2, to train the perceptron.

Implement two functions:

1. a testing function, `predictLinearModel` that returns the predictions of a model on a test data set
2. an evaluation function, `evaluateLinearModel`, that computes the accuracy of the model on the test data by calculating the fraction of observations for which the predicted label is same as the true label.

REPORT 3.
Train the perceptron model by calling the `scipy.optimize.minimize` method and use the `evaluateLinearModel` to calculate and report the accuracy for the training and test data.

Problem 4: Using Newton's Method for Logistic Regression Learning (20 code + 5 report = 25 points)

For this problem, you will train a logistic regression model, whose loss function (also known as the *logistic-loss* or *log-loss*) is given by:

$$J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\log\left(1 + \exp\left(-y_i \mathbf{w}^\top \mathbf{x}_i\right)\right) \tag{9}$$

[1]For Logistic Regression, typically a different formulation is presented. The decision rule is written as:

$$y_i = \begin{cases} -1 & \text{if } \theta_i < 0.5 \\ +1 & \text{if } \theta_i \geq 0.5 \end{cases} \tag{6}$$

where,

$$\theta_i = \sigma(\mathbf{w}^\top \mathbf{x}_i) = \frac{1}{1 + \exp -(\mathbf{w}^\top \mathbf{x}_i)} \tag{7}$$

However, one can see that it is equivalent to checking if $\mathbf{w}^\top \mathbf{x}_i < 0$ or not.

The gradient for this loss function is given by, as derived in the class:

$$\nabla J(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}\frac{y_i}{1+\exp\left(y_i\mathbf{w}^\top\mathbf{x}_i\right)}\mathbf{x}_i \tag{10}$$

The *Hessian* for the loss function is given by:

$$\mathbf{H}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\frac{\exp\left(y_i\mathbf{w}^\top\mathbf{x}_i\right)}{\left(1+\exp\left(y_i\mathbf{w}^\top\mathbf{x}_i\right)\right)^2}\mathbf{x}_i\mathbf{x}_i^\top \tag{11}$$

Newton's Method The update rule is given by:

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} + \eta\mathbf{H}^{-1}(\mathbf{w}^{(t-1)})\nabla J(\mathbf{w}^{(t-1)})$$

However, for this assignment we will be using the `scipy.optimize.minimize` function again, with `method = 'Newton-CG'`, for training using the Newton's method. This will need you to implement the following three functions:

1. `logisticObjVal` - compute the logistic loss for the given data set (See (9)).
2. `logisticGradient` - compute the gradient vector of logistic loss for the given data set (See (10)).
3. `logisticHessian` - compute the Hessian matrix of logistic loss for the given data set (See (11)).

REPORT 4.
Train the logistic regression model by calling the `scipy.optimize.minimize` method, and use the `evaluateLinearModel` to calculate and report the accuracy for the training and test data.

Problem 5: Using Stochastic Gradient Descent Method for Training Linear Support Vector Machine (20 code + 5 report = 25 points)

While we will study the quadratic optimization formulation for SVMs in class, we can also train the SVM directly using the *hinge-loss* given by:

$$J(\mathbf{w}) = \sum_{i=1}^{n}\max\left(0, 1-y_i\mathbf{w}^\top\mathbf{x}_i\right) \tag{12}$$

Clearly, the above function is not as easily differentiable as the *squared-loss* and *logistic-loss* functions above, we can devise a simple *Stochastic Gradient Descent* (SGD) based method for learning $\mathbf{w}$. Note that, for a single observation, the loss is given by:

$$J_i(\mathbf{w}) = \max\left(0, 1-y_i\mathbf{w}^\top\mathbf{x}_i\right) \tag{13}$$

$$= \begin{cases} 1-y_i\mathbf{w}^\top\mathbf{x}_i & \text{if } y_i\mathbf{w}^\top\mathbf{x}_i < 1 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Thus, the gradient of $J_i(\mathbf{w})$ can be written as:

$$\nabla J_i(\mathbf{w}) = \begin{cases} -y_i\mathbf{x}_i & \text{if } y_i\mathbf{w}^\top\mathbf{x}_i < 1 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

The training can be done using the following algorithm:

1: $\mathbf{w} \leftarrow [0, 0, \ldots, 0]^\top$
2: **for** t=1, 2, ... T **do**

3: $i \leftarrow RandomSample(1 \ldots n)$
4: **if** $y_i \mathbf{w}^\top \mathbf{x}^{(i)} < 1$ **then**
5: $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$
6: **end if**
7: **end for**

You have to implement a function `trainSGDSVM` that learns the optimal weight, $\mathbf{w}$ using the above algorithm.

REPORT 5.

Train the SVM model by calling the `trainSGDSVM` method for 200 iterations (set learning rate parameter η to 0.01). Use the `evaluateLinearModel` to calculate and report the accuracy for the training and test data.

Problem 6: Comparing Linear Classifiers (0 code + 5 report = 5 points)

Using the results for Problems 3–4, provide a comparison of the three different linear models (Perceptrons, Logistic Regression, and Support Vector Machines) on the provided data set.

REPORT 6.

1. Use the results for test data to determine which classifier is the most accurate?
2. Plot the decision boundaries learnt by each classifier using the provided `plotDecisionBoundary` function which takes the learnt weight vector, $\mathbf{w}$ as one of the parameters. Study the three boundaries and provide your insights.
