Korea Advanced Institute of Science and Technology
School of Electrical Engineering
EE488 Intro. to Machine Learning, Spring 2018

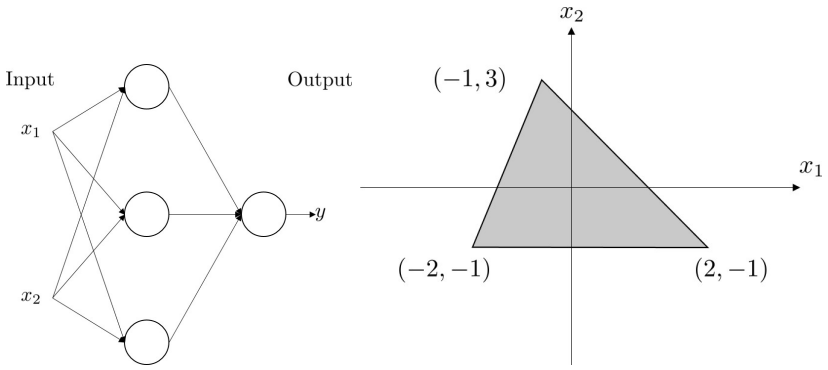
Issued: Mar. 5, 2018
Due: Mar. 21, 2018

Assignment I

Policy

Group study is encouraged; however, assignment that you hand-in must be of your own work. Anyone suspected of copying others will be penalized. The homework will take considerable amount of time so start early.

1. **Multi-layer perceptron**
   The function of a perceptron can be seen as to divide the feature space into two half-spaces. Consider a multi-layer perceptron with each neuron having threshold activation function. Multi-layer perceptron is considered as shown below with the threshold activation function. There are two inputs x_1 and x_2. Each neuron also has a bias input. Determine the weights manually so that the decision region is such that the output is 1 (unity) in the shaded area and 0 (zero) outside the shaded area as shown in the figure below. Let $f(u) = 1$ if $u > 0$, otherwise 0.



2. **Robust Linear Regression**
   Assume that there are n given training examples $(x_1, y_1), \ldots, (x_N, y_N)$, where each input data point $x_i \in \mathbb{R}^m$. The goal of regression is to learn to predict y from x. The linear regression model assumes that the output y is a linear combination of the input features x plus noise terms ϵ from a given distribution with weights given by θ. We can write this in matrix form by stacking the datapoints as the rows of a matrix X so that $\phi_j(x_i)$ is the j-th feature of the i-th datapoint. Then writing Y , θ and ϵ as column vectors, we can write the matrix form of the linear regression model as:

$$Y = X\theta + \boldsymbol{\epsilon}$$

   where

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_N \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_p \end{bmatrix}, \text{ and } X = \begin{bmatrix} \phi_0(x_1) & \phi_1(x_1) & \ldots & \phi_p(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \ldots & \phi_p(x_N) \end{bmatrix}$$

Linear regression seeks to find the parameter vector θ that provides the best fit of the above regression model. One criteria to measure fitness, is to find θ that minimizes a given loss function such as the loss function to be the square-error.

$$\mathcal{L}(\theta) = (Y - X\theta)^T(Y - X\theta)$$

(i) Assume $\{\epsilon_i\}_{i=1}^n$ are independent identical distributed(i.i.d.) with $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \sigma^2 \boldsymbol{I})$. Derive the maximum likelihood estimate of θ in other words $\hat{\theta} = \arg\max_\theta \prod_{i=1} \mathcal{N}(\epsilon_i; 0, \sigma^2)$.

(ii) This time assume $\{\epsilon_i\}_{i=1}^n$ are independent identical distributed(i.i.d.) with $\epsilon_i \sim \mathit{Laplace}(\epsilon; 0, b)$. Derive the maximum likelihood estimate of θ in other words $\hat{\theta} = \arg\max_\theta \prod_{i=1} \mathit{Laplace}(\epsilon_i; 0, b)$. Here $\mathit{Laplace}(\epsilon; \mu, b) = \frac{1}{2b}\exp(-\frac{|\epsilon-\mu|}{b})$, where μ is the location parameter, and b is the scale parameter.

The square error loss weights each observation equally irrespective of the magnitude of the error. This is may not be appropriate for handling outliers which produce large residuals. What may be more appropriate is to downplay the outliers and model the noise with a Laplace distribution instead of a normal distribution.

(iii) Why do you think that the above model provides a more robust fit to data compared to the standard model assuming Gaussian distribution of the noise terms?

3. **SVM Error Analysis**

In this problem, we want to analyze the error of SVM classification. Assume that we have n data points $(\mathbf{x}_i, y_i)_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^m$, $y_i \in [k] = 1, 2 \ldots k$. Let $f_{(\mathbf{x}_1,y_1),\cdots,(\mathbf{x}_n,y_n)}$ denote SVM classifier trained on these n data points.

For a randomly drawn test data point $(\mathbf{x}_{n+1}, y_{n+1})$, the prediction is $y_{n+1}^{\text{pred}} = f_{(\mathbf{x}_1,y_1),\cdots,(\mathbf{x}_n,y_n)}(\mathbf{x}_{n+1})$. We assume that the n training data points and the test data point $(\mathbf{x}_{n+1}, y_{n+1})$ are drawn i.i.d from some unknown underlying distribution. The expected error rate is defined as:

$$\texttt{err} = \mathbb{E}_{(\mathbf{x}_1,y_1),\cdots,(\mathbf{x}_{n+1},y_{n+1})} \mathbb{1}\{f_{(\mathbf{x}_1,y_1),\cdots,(\mathbf{x}_n,y_n)}(\mathbf{x}_{n+1}) \neq y_{n+1}\}$$

where the indicator function $\mathbb{1}\{A\} = 1$ if A is true; otherwise 0.

(i) Show that the expected error rate is equal to the expectation of leave-one-out cross validation(LOOCV) error for $n + 1$ data points. In LOOCV, each of the n different models is trained excluding one of n training data (training data size of $(n-1)$), and each is tested on the excluded data. Performance is obtained as the mean test performance of n models.

(ii) "The leave-one-out cross-validation error does not depend on the dimensionality of the feature space but only on the number of support vectors." Show that this statement is true by explaining why

$$\texttt{err}_{\text{LOOCV}} \leq \frac{n_s}{n+1}$$

where $\texttt{err}_{\text{LOOCV}}$ is the leave-on-out cross-validation error for training set $(\mathbf{x}_i, y_i)_{i=1}^{n+1}$, n_s is the number of support vectors.

4. **Gradient Descent**

Gradient Descent can be thought of as updating $\theta^{(t)}$ with $\theta^{(t+1)}$ which is near and jointly minimizing the loss function. The loss function is approximated as the first-order Taylor series expansion of $\mathcal{L}(\theta)$ near $\theta^{(t)}$. This can be written mathematically as $\theta^{(t+1)} = \arg\min_\theta \frac{1}{2}||\theta - \theta^{(t)}||^2 + \eta(\mathcal{L}(\theta^{(t)}) + < \theta - \theta^{(t)}, \nabla\mathcal{L}(\theta^{(t)}) >)$. Show that this minimization leads to gradient descent.

5. **Perceptron & SVM Algorithm(Matlab Programming Assignment)**

(i) **Implementing Perceptron algorithm**

In this section, we will implement a perceptron algorithm. Before studying SVM, a learning algorithm is referred to as perceptron was introduced. It is a linear classifier given as

$$y = \text{sgn}(\theta^\top \mathbf{x} + \theta_0) = \begin{cases} 1 & \text{if } \theta^\top \mathbf{x} + \theta_0 \geq 0, \\ -1 & \text{if } \theta^\top \mathbf{x} + \theta_0 < 0. \end{cases}$$

As we have studied in class, the parameters of the classifier are only updated for misclassified training examples. Batch-based perceptron uses stochastic gradient descent(SGD) to minimize the sum of the functional margins of misclassified examples such that

$$\mathcal{L}(\theta, \theta_0) = -\sum_{i \in \mathcal{M}} y_i(\theta^T \mathbf{x}_i + \theta_0)$$

$$\frac{\partial \mathcal{L}(\theta, \theta_0)}{\partial \theta} = -\sum_{i \in \mathcal{M}} y_i \mathbf{x}_i \qquad \frac{\partial \mathcal{L}(\theta, \theta_0)}{\partial \theta_0} = -\sum_{i \in \mathcal{M}} y_i$$

For implementing the perceptron algorithm, a skeleton code is provided in the "perceptron" folder. The main function `main_perceptron.m` calls the following 4 functions: (1) `[x_train,y_train,x_test,y_test] = createDataset('train.csv','test.csv')`, (2) `[theta] = train(x_train,y_train,T)`, (3) `[acc] = test(x_test,y_test,theta)` and (4) `plot_perceptron(x_train,y_train,x_test,y_test,theta)`.

a. `[x_train,y_train,x_test,y_test] = createDataset('train.csv','test.csv')` loads both the training and test dataset contained respectively in `train.csv` and `test.csv`.

b. `[theta] = train(x_train,y_train,eta,T)` takes both the training set {`x_train,y_train`} where `x_train` is the data array while `y_train` is the corresponding label array of training dataset and learning rate(η) `eta` referred to as step size or learning rate as inputs to the perceptron and outputs a weight vector `theta`.

c. `[acc]=test(x_test,y_test,theta)` takes test set {`x_test,y_test`}, the estimated weight vector `theta` and estimates the classification accuracy in `acc`.

d. `plot_perceptron(x_train,y_train,x_test,y_test,theta)` takes the training set {`x_train,y_train`} to scatter plots the loaded data and makes the use of the trained weight vector `theta` to draw a decision boundary on the scatter plot. The positive samples should be represented as red dots while the negative samples should be in blue dots.

(ii) **Implementing Soft-SVM Using SGD**

In this problem, we implement a very simple algorithm for solving the optimization problem of Soft-SVM, namely,

$$\min_\theta \Big(\frac{\lambda}{2}||\theta||^2 + \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i\langle\theta, \mathbf{x}_i\rangle\}\Big) = \min_\theta \Big(\frac{\lambda}{2}||\theta||^2 + \mathcal{L}_s(\theta)\Big).$$

We can solve this problem using SGD framework whose update rule is defined as:

$$\theta^{(t+1)} = -\frac{1}{\lambda t}\sum_{j=1}^{t} \mathbf{v}_j,$$

where $\mathbf{v}_j$ is a subgradient of the loss function($\mathcal{L}_s$) at $\theta^{(j)}$ on random example chosen at iteration j. For the hinge loss, given an example $(\mathbf{x}, y)$, we can choose $\mathbf{v}_j$ to be 0 when

$y\langle\theta^{(j)}, \mathbf{x}\rangle \geq 1$; otherwise $\mathbf{v}_j = -y\mathbf{x}$. Denoting $\mathbf{w}^{(t)} = -\sum_{j<t} \mathbf{v}_j$ we obtain the following procedure.

Algorithm 1: SGD for solving Soft-SVM

Goal : Solve Equation
Parameter : T
Initialization: $\mathbf{w}^{(1)} = 0$
for $t = 1, ..., T$ **do**
  Let $\theta^{(t)} = \frac{1}{\lambda t}\mathbf{w}^{(t)}$
  Choose i uniformly at random from $[m]$(m is the number of data)
  if $(y_i\langle\theta^{(t)}, \mathbf{x}_i\rangle) < 1$ **then**
    Set $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i\mathbf{x}_i$
  else
    Set $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)}$
  end
end
Output : $\bar{\theta} = \frac{1}{T}\sum_{t=1}^{T} \theta^{(t)}$

For implementing the svm algorithm, a skeleton code is provided in the "svm" folder. The main function `main_svm.m` calls the following 4 functions: (1) `[x_train,y_train,x_test,y_test] = createDataset('train.csv','test.csv')`, (2) `[theta], = train(x_train,y_train,T)`, (3) `[acc] = test(x_test,y_test,theta)` and (4) `plot_svm(x_train,y_train,x_test,y_test,theta)`.

a. `[x_train,y_train,x_test,y_test] = createDataset('train.csv','test.csv')` loads both the training and test dataset contained respectively in `train.csv` and `test.csv`.
b. `[theta] = train(x_train,y_train,T)` takes both the training set {`x_train,y_train`} where `x_train` is the data array while `y_train` is the corresponding label array of training dataset as inputs to the svm and outputs a weight vector `theta`.
c. `[acc]=test(x_test,y_test,theta)` takes test set {`x_test,y_test`}, the estimated weight vector `theta` and estimates the classification accuracy in `acc`.
d. `plot_svm(x_train,y_train,x_test,y_test,theta)` takes the training set {`x_train,y_train`} to scatter plots the loaded data and makes the use of the trained weight vector `theta` to draw a decision boundary on the scatter plot. The positive samples should be represented as red dots while the negative samples should be in blue dots.

Submit Instructions for Programming Assignment

Please submit in .zip file to KLMS named "ee488_assignment1_student#.zip" , for example, "ee488_assignment1_20181234.zip".
This file should contain two folders and one document file for plotted result and explanation of the result.
In python code, the comment explaining your code **must be** included, or you will not get a full grade even if your code works fine. Please also include all the files that are required to run the code in the zip file. Do not change the name of the folder and comments should be written in English. Additionally submitting unexecutable code will receive no points. Using other libraries such as 'scipy' are not allowed.