CS 8725: Report for assignment 3

Chanmann Lim

September 28, 2015

The Matlab code for all experiments is in the **Appendix** section.

Programming 1: We are given a bunch of data samples of "Average height and weight of American women aged 30 to 39" and the task is to design linear regression algorithm to predict the *weight* $\in \mathbb{R}$ denoted by Y from a given height measurement denoted by X then the problem becomes $f(X) \rightarrow Y$ finding a function mapping from X to Y. In linear regression we assume that $f(X)$ take a linear form with respect to X that is $f(X) = \beta_1 + \beta_2 X$ and the goal is to choose $\hat{f}(X)$ that minimizes the prediction error (squared error).

$$\hat{f}_n^L = \underset{f}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (f(X_i) - Y_i)^2 \tag{1}$$

By rewriting $f(X)$ in term of $\beta = [\beta_1 \ \beta_2]^T$, minimizing $f(X)$ becoming minimizing β.

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (X_i\beta - Y_i)^2 \tag{2}$$

$$= \underset{\beta}{\operatorname{argmin}} \frac{1}{n} (A\beta - Y)^T (A\beta - Y) \tag{3}$$

Where,

$$A = \begin{bmatrix} 1 & X_1 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix} \text{ and } Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} \tag{4}$$

If we define $J(\beta) = (A\beta - Y)^T (A\beta - Y)$ then minimizing $J(\beta)$ is equivalent to (3).

$$\frac{\partial J(\beta)}{\partial \beta} = \frac{\partial A^T A \beta \beta^T - 2\beta^T A^T Y + Y^T Y}{\partial \beta} \tag{5}$$

$$= 2A^T A\beta - 2A^T Y \tag{6}$$

$$2A^T A\hat{\beta} - 2A^T Y = 0 \tag{7}$$

$$A^T A\hat{\beta} = A^T Y \tag{8}$$

$$\hat{\beta} = (A^T A)^{-1} A^T Y \tag{9}$$

Eq. (9) is the normal equation with $(A^T A)$ as normal matrix and $\hat{f}_n^L = X\hat{\beta}$. As a result we obtained the linear regression coefficient in (10) and the expected error in (13):

$$\hat{\beta} = [-39.0620 \quad 61.2722]^T \tag{10}$$

$$R(\hat{f}) = \mathbb{E}[(\hat{f}(X) - Y)^2] \tag{11}$$
$$= \frac{1}{n}(X\hat{\beta} - Y)^T(X\hat{\beta} - Y) \tag{12}$$
$$= 0.4994 \tag{13}$$

In fact for this case in particular with the hint from the lecture note, we know that we could further reduce the expected error $R(\hat{f})$ by introducing quadratic term into the regression. With this procedure we have turned linear regression into polynomial regression and by following the derivation in solving for $\hat{\beta}$ it turn out that $\hat{\beta} = (A^T A)^{-1} A^T Y$ where $\beta = [\beta_1\ \beta_1\ \beta_3]^T$ and

$$A = \begin{bmatrix} 1 & X_1 & X_1^2 \\ \vdots & \vdots & \vdots \\ 1 & X_n & X_n^2 \end{bmatrix}$$

We obtained $\hat{\beta} = [128.8128 \quad -143.1620 \quad 61.9603]^T$ and $R(\hat{f}) = 0.0506$ for second order polynomial regression. Figure 1 show the both linear and polynomial regression functions with respect to the data samples.

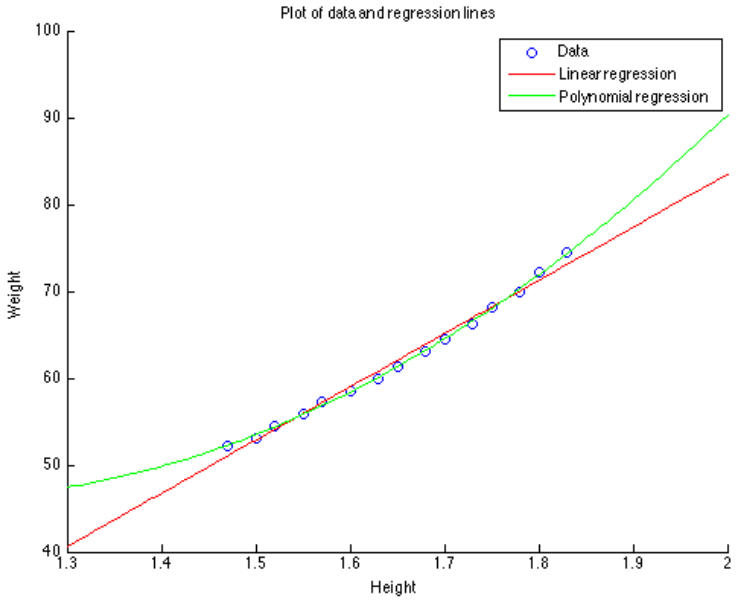

Figure 1: Plot of data and regression lines

Programming 2: In this task we are going to implement logistic regression classifier using iterative algorithm namely **gradient ascent** and test it on the Iris plant dataset by R.A Fisher available at UCI machine learning repository(`http://archive.ics.uci.edu/ml/datasets/Iris`). The dataset consists of four features such as *sepal length in cm*, *sepal width in cm*, *petal length in cm* and *petal width in cm* and the class label. There are 50 instances from each class which make it a total of 150 samples however we are only interested in binary classification of *Iris-Setosa* and *Iris-Versicolor*, the other class *"Iris-Virginica"* were removed from the dataset in the pre-processing phase. In addition we understand that *Iris-Setosa* class is linearly separable from *Iris-Versicolor*.

In logistic regression, instead of trying to learn the likelihood probability $P(X|Y)$ it aims at learning $P(Y|X)$ the conditional likelihood directly from the data. And for binary classification the assumption is that:

$$P(Y=0|X,w) = \frac{1}{1+exp(w_0+\sum_{i=1}^{d} w_i X_i)} \tag{14}$$

$$P(Y=1|X,w) = \frac{exp(w_0+\sum_{i=1}^{d} w_i X_i)}{1+exp(w_0+\sum_{i=1}^{d} w_i X_i)} \tag{15}$$

Henceforth, the job is find the $\hat{\mathbf{w}}$ that maximizes the posterior probability.

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{j=1}^{n} P(Y^{(j)}|X^{(j)}, \mathbf{w}) \tag{16}$$

From i.i.d assumption we have $p(D|\mathbf{w}) = \prod_{j=1}^{n} p(y^j|x^j, \mathbf{w})$. Since $y \in \{0,1\}$ we can rewrite $p(D|\mathbf{w})$ as:

$$p(D|\mathbf{w}) = \prod_{j=1}^{n} p(y^j=1|x^j,\mathbf{w})^{y^j} p(y^j=0|x^j,\mathbf{w})^{1-y^j} \tag{17}$$

$$= \prod_{j=1}^{n} \left(\frac{exp(w_0+\sum_i w_i x_i^j)}{1+exp(w_0+\sum_i w_i x_i^j)}\right)^{y^j} \left(\frac{1}{1+exp(w_0+\sum_i w_i x_i^j)}\right)^{1-y^j} \tag{18}$$

$$l(\mathbf{w}) = ln\ p(D|\mathbf{w}) = \sum_{j=1}^{n} \left[y^j (w_0 + \sum_{i}^{d} w_i x_i^j) - ln(1+exp(w_0+\sum_{i}^{d} w_i x_i^j)) \right] \tag{19}$$

Then maximizing $l(\mathbf{w})$ the conditional log likelihood with respect to all elements of $\mathbf{w}$.

$$\frac{\partial l(\mathbf{w})}{\partial w_i} = \sum_{j=1}^{n} \left[y^j x_i^j - \frac{exp(w_0+\sum_i w_i x_i^j)}{1+exp(w_0+\sum_i w_i x_i^j)} x_i^j \right] \tag{20}$$

$$= \sum_{j=1}^{n} \left[x_i^j (y^j - p(y^j=1|x_i^j,\mathbf{w}) \right] \tag{21}$$

The issue here is that there is no closed-form solution to solve for $\mathbf{w}$ directly, yet $l(\mathbf{w})$ is a concave function which will lead to the convergence at the global maximum by applying any optimization technique such as **gradient ascent** or newton's methods, etc,.

Gradient Ascent: we started by initializing $\mathbf{w} = [w_0\ w_1\ w_2\ w_3\ w_4\ w_5]^T$ randomly between [-1, 1] then iteratively compute the gradient $\nabla_{\mathbf{w}} l(\mathbf{w}) = \left[\frac{\partial l(\mathbf{w})}{\partial w_0}, \cdots, \frac{\partial l(\mathbf{w})}{\partial w_5}\right]^T$ and update $\mathbf{w}$ until a fixed number of iterations $T = 100000$ or stop when it converged.

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \alpha \frac{\partial l(\mathbf{w})}{\partial w_i} \tag{22}$$

$$= w_i^{(t)} + \alpha \sum_{j=1}^{n} x_i^j (y^j - p(y^j=1|x_i^j,\mathbf{w}) \tag{23}$$

Where $w_i^{(t)}$ is the weight for i^{th} component of $\mathbf{w}$ at iteration t and α is the learning rate for which we chose $\alpha = 0.001$. And here we defined the convergence criteria as the percentage change of the conditional log likelihood $l(\mathbf{w})$ to be less then 0.03% ($\epsilon = 0.0003$).

$$(\Delta\% l(\mathbf{w}) < \epsilon) \rightarrow \text{converge} \tag{24}$$

We can see in Figure 2. that the conditional log likelihood scores keep increasing and **gradient ascent** algorithm converged at $t = 3022$ for:

$$\mathbf{w}^{(t=0)} = [-0.0665, -0.4920, -0.1376, 0.4051, -0.1953]^T \tag{25}$$

$$\mathbf{w}^{(t=3022)} = [-0.4187, -0.8509, -2.3051, 4.0195, 1.4223]^T \tag{26}$$

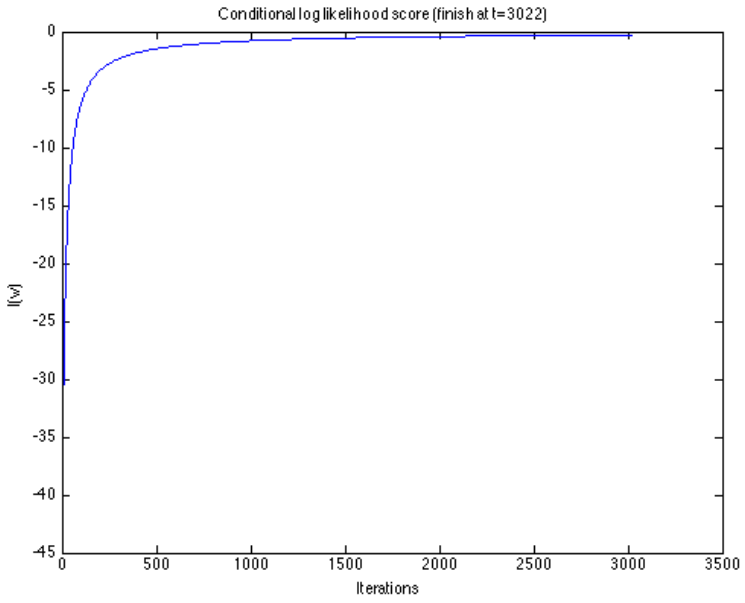

Figure 2: Plot of conditional log likelihood scores

To evaluate, we can compare and predict the class with higher posterior probability between $P(Y = 0|X, \mathbf{w})$ and $P(Y = 1|X, \mathbf{w})$ and decision boundary is

$$\frac{P(Y = 1|X, \mathbf{w})}{P(Y = 0|X, \mathbf{w})} = 1 \tag{27}$$

$$exp(w_0 + \sum_i^d w_i X_i) = 1 \tag{28}$$

$$w_0 + \sum_i^d w_i X_i = 0 \tag{29}$$

At this stage we understand that the decision boundary for logistic regression is simply a linear function and since the dataset we are working with for this experiment is linearly separable we expect our classifier to be able to perfectly predict the correct category of each samples. And the outcome yield 100% training accuracy which is indeed consistent with our presumption.

$$accuracy = \frac{\sum_i^n \mathbf{1}_{predicted_i = Y_i}}{n} \tag{30}$$

$$= 1 \tag{31}$$

Experimental:

Choice of $\mathbf{w}^{(t=0)} = \mathbf{0}$ is not at all bad of a choice since we know that gradient ascent will converge due to the concavity of the function we are maximizing however good practices in general optimization procedure avoid this since it starts the optimization at $P(Y = 0|X) = P(Y = 1|X) = 0.5$ thus might be more likely to lead to local optima and get stuck than a randomly generated one.

Choice of α, the learning rate or step-size in gradient ascent algorithm, is used to control the penalized contribution of a particular $\mathbf{w}$ at each iteration. Although there is no rule of thumb in determining the right value for α other than the constraint that $\alpha \in [0, 1]$, empirical successes suggested that given large enough number of iterations we could start with $\alpha = 1$ then keep tuning by dividing the value of α by three (eg. $\alpha = \{1, 0.3, 0.1, 0.03, 0.01, \cdots\}$) and we found that $\alpha = 0.001$ give us smooth convergence comparing to other numbers. While smaller α make gradient ascent stop because either the change of $l(\mathbf{w})$ is not significant or it reached the pre-defined maximum iterations $T = 100000$; larger α oscillated the value of $l(\mathbf{w})$ which we are trying to maximize if not diverge completely.

Appendix:

assignment_3.m

```
clc;
clear all;
close all;

problem_1
problem_2
```

problem_1.m

```
% Load data
x = [1.47 1.5 1.52 1.55 1.57 1.6 1.63 1.65 1.68 1.7 1.73 1.75 1.78 1.8 1.83]';
y = [52.21 53.12 54.48 55.84 57.2 58.57 59.93 61.29 63.11 64.47 66.28 68.1 69.92 72.19 74.46]';

% Linear regression
A = [ones(length(x), 1) x];
Beta = (A'*A)\A'*y;
display(Beta);

% Evaluation
prediction = A*Beta;
expected_loss_beta = (y-prediction)' * (y-prediction) / length(y);
display(expected_loss_beta);

% Polynomial regression
A = [A x.^2];
Beta_2 = (A'*A)\A'*y;
display(Beta_2);

% Evaluation
prediction = A*Beta_2;
expected_loss_beta2 = (y-prediction)' * (y-prediction) / length(y);
display(expected_loss_beta2);

% Plot
figure;
scatter(x, y); hold on;
x_value = linspace(1.3, 2);
plot(x_value, Beta(1) + Beta(2)*x_value, 'r');
plot(x_value, Beta_2(1) + Beta_2(2)*x_value + Beta_2(3)*x_value.^2, 'g');
hold off;
title('Plot of data and regression lines');
xlabel('Height');
ylabel('Weight');
legend('Data', 'Linear regression', 'Polynomial regression');
```

problem_2.m

```
% Load data
[x1,x2,x3,x4] = textread('iris.data', '%f,%f,%f,%f,%*s');
X = [ones(length(x1),1) x1 x2 x3 x4];

setosa = 1:50;
versicolor = 51:100;
[~, d] = size(X);
y = [zeros(length(setosa),1); ones(length(versicolor),1)];

% Logistic regression gradient ascent
% initialize weight = [-1, 1]
W = 2*rand(d, 1)-1;
display(W);
% maximum iteration
T = 100000;
% learning rate
alpha = 0.001;
% conditional log likelihood
lw = zeros(1, T);

for t=1:T
    prediction = exp(X*W) ./ (1+exp(X*W));
```

```
    W = W + alpha * X' * (y - prediction);
    lw(t) = y'*X*W - sum( log(1+exp(X*W)) );
    if t > 1
        delta_lw = abs( (lw(t) - lw(t-1)) / lw(t-1) );
        % convergence criterion = percentage change < 0.03%
        if delta_lw < 3e-4
            break;
        end
    end
end
display(W);

% Evaluation
classification = X*W > 0;
accuracy = sum(classification == y) / length(y);
display(accuracy);

% Plot of conditional log likelihood
figure;
plot(1:t, lw(1:t));
title(['Conditional log likelihood score (finish at t=' num2str(t) ')']);
xlabel('Iterations');
ylabel('l(w)');
```