

School of Computing and Information Systems
The University of Melbourne
COMP90049 Introduction to Machine Learning (Semester 1, 2023)
Week 9 – Sample Solution

1. What is Logistic Regression? What is “logistic”? What are we “regressing”?

We build a Binary Logistic Regression model, where the target (y) is (close to) 1 for instances of the positive class, and (close to) 0 for instance of the negative class. (Suitably can be re-interpreted for multi-class problems.)

The goal of binary logistic regression is to train a classifier that can make a binary decision about the class of an input observation. Consider a test instance X , which we will represent by a vector of features $[x_1, x_2, \dots, x_n]$. The output y can be 1 or 0 (i.e., winning / not winning).

The model calculates the probability $P(y=1|x)$ (from which we can trivially derive $P(y=0|x)$). A logistic regression classifier additionally defines a ‘decision boundary’, which is typically set at 0.5. If the model predicts the probability $P(Y=1|x) > 0.5$, we classify x as class 1. Otherwise, x is classified as class 0.

In statistics, regression describes how an independent variable is numerically related to the dependent variable. In Logistic Regression we find the regression output z ($z = \vec{\theta} \cdot \vec{X} = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$), where θ is the weight of the features. θ represents the numeric correlation between each feature (attribute) with the label (class), where the feature (attributes) of our dataset is assumed as our independent variables and the label (class) is our dependent variable.

Since Logistic (or sigmoid) function has an easy-to-calculate derivative (that makes it easy to calculate the optimum parameters), and has a range of $[0, 1]$, we apply the logistic function $\sigma = \frac{1}{1+e^{-z}}$ to the regression output z and call it Logistic Regression.

2. In following dataset each instance represents a news article. The value of the features are counts of selected words in each article. Develop a logistic regression classifier to predict the class of the article (fruit vs. computer). $\hat{y} = 1$ (fruit) and $\hat{y} = 0$ (computer).

ID	<i>apple</i>	<i>ibm</i>	<i>lemon</i>	<i>sun</i>	CLASS	
TRAINING INSTANCES						
A	1	0	1	5	1	FRUIT
B	1	0	1	2	1	FRUIT
C	2	0	0	1	1	FRUIT
D	2	2	0	0	0	COMPUTER
E	1	2	1	7	0	COMPUTER
TEST INSTANCES						
<i>T</i>	1	2	1	5	0	COMPUTER

For the moment, we assume that we already have an estimate of the model parameters, i.e., the weight of the 4 features (and the bias θ_0) is $\hat{\theta} = [\theta_0, \theta_1, \theta_2, \theta_3, \theta_4] = [0.2, 0.3, -2.2, 3.3, -0.2]$.

(i). Explain the intuition behind the model parameters in relation to the features.

In this dataset we want identify if a piece of writing is about computer or fruit (e.g. ‘new apple iPhone is very expensive’ vs. ‘an apple a day, keeps the doctor away’). To do so, we are using 4 terms (apple, ibm, lemon, sun) and the count of their occurrences in a piece of writing. So, for example we know that Doc A includes apple once and sun five times.

The decision to include exactly these for terms (and no others) as attributes, and to define their values

as word occurrence counts is the decision of the modeler, and the outcome of a process called **Feature Engineering**.

Based on the definition, we know that in Logistic Regression, we model $P(y = 1 | x_1, x_2, \dots, x_F)$ directly as subject to parameter θ . Using the following:

$$P(y = 1 | x_1, x_2, \dots, x_F) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \dots + \theta_F x_F)}} = \sigma(\theta_0 + \theta_1 x_1 + \dots + \theta_F x_F)$$

Here, we have a binary output label $y = \{0 (\text{computer}), 1 (\text{fruit})\}$, and four features: apple, ibm, lemon, sun.

Weights $\theta_1, \theta_2, \theta_3, \theta_4$ denote the respective importance of the features 1 to 4 for predicting the class fruit (1). For example, θ_2 (-2.2) indicates how important feature 2 (ibm) is for predicting $\hat{y} = 1$ (fruit). θ_0 represent the bias for this model.

Here, the negative sign indicates that occurrence of the word ibm is *negatively correlated* with the class FRUIT. If we observe ibm in a document, it makes the label FRUIT *less likely*. The positive value of θ_3 indicates a *positive correlation* of feature lemon with FRUIT. If the word lemon is mentioned in a document, it *increases the probability* of its label being FRUIT.

(ii). Predict the test label.

Using the logistic regression formula above, we find

$$\begin{aligned} P(\text{fruit}|T) &= P(\hat{y} = 1|T) = \sigma(\theta_0 + \theta_1 t_1 + \dots + \theta_4 t_4) \\ &= \sigma(0.2 + 0.3 \times 1 + (-2.2) \times 2 + 3.3 \times 1 + (-0.2) \times 5) \\ &= \sigma(-1.6) = \frac{1}{1 + e^{-(-1.6)}} = 0.17 \end{aligned}$$

And consequently

$$P(\text{computer}|T) = 1 - P(\text{fruit}|T) = 1 - 0.17 = 0.83$$

Recall that we turn the logistic regression model into a classifier by predicting label $y = 1$ whenever $p(y=1|x, \theta) > 0.5$, and predict $y = 0$ otherwise.

Since the for the test instance T the probability $p(y=1)$ (FRUIT) is smaller than 0.5, we predict label $y=0$ (COMPUTER) for T. Comparing with our “ground truth” that we have in our dataset, we can see that our prediction is correct. ☺

(iii). Recall the conditional likelihood objective

$$-\log \mathcal{L}(\theta) = -\sum_{i=1}^n y_i \log(\sigma(x_i; \theta)) + (1 - y_i) \log(1 - \sigma(x_i; \theta))$$

Design a test to make sure that the Loss of our model, is lower when its prediction the correct label for test instance T, than when it's predicting a wrong label.

To test our error function, we are going to compute the negative log-likelihood of the test instance (1) assuming the true label as $y = 1$ (fruit), i.e., our classifier made a mistake; and (2) assuming that the true label $y = 0$ (computer), i.e., our classifier predicted correctly.

- Assuming $y = 1$ and we predicted $\hat{y} = 0$:

$$\begin{aligned}
-\log \mathcal{L}(\theta) &= -\sum_{i=1}^n 1 \log(\sigma(0.2 + 0.3 \times 1 + (-2.2) \times 2 + 3.3 \times 1 + (-0.2) \times 5)) \\
&\quad + (1 - 1) \log(1 - \sigma(0.2 + 0.3 \times 1 + (-2.2) \times 2 + 3.3 \times 1 + (-0.2) \times 5)) \\
&= -\sum_{i=1}^n 1 \log(\sigma(-1.6)) + (1 - 1) \log(1 - \sigma(-1.6)) \\
&= -[1 \log(\sigma(-1.6)) + 0] = -\log \frac{1}{1 + e^{-(-1.6)}} = -\log(0.17) = 1.77
\end{aligned}$$

- Assuming $y = 0$ and we predicted $\hat{y} = 0$:

$$\begin{aligned}
-\log \mathcal{L}(\theta) &= -\sum_{i=1}^n 0 \log(\sigma(-1.6)) + (1 - 0) \log(1 - \sigma(-1.6)) \\
&= -[0 + 1 \log(1 - \sigma(-1.6))] = -\log(1 - 0.17) = -\log(0.83) = 0.19
\end{aligned}$$

Reassuringly, the negative log likelihood (aka loss) under predicted label $y = 0$ (correct) is **lower** than the loss for label $= 1$ which is the incorrect prediction.

- For the model created in question 2, compute a single gradient descent update for parameter θ_1 given the training instances given above. Recall that for each feature j , we compute its weight update as

$$\theta_j \leftarrow \theta_j - \eta \sum_i (\sigma(x_i; \theta_i) - y_i) x_{ij}$$

Summing over all training instances i . We will compute the update for θ_j assuming the current parameters as specified above, and a learning rate $\eta = 0.1$.

θ_1 is the model parameter (respective importance) of the features 1 (apple) in our model. Using Gradient Descent method over iterations, we want to find the best parameters of the model (the parameters that minimize the loss (error) of our model).

Step 1, we need to compute the $\sigma(x_i; \theta)$ for all our training instances.

$$\sigma(x_A; \theta) = \sigma(0.2 + (0.3 \times 1 + (-2.2) \times 0 + 3.3 \times 1 + (-0.2) \times 5)) = 0.94$$

$$\sigma(x_B; \theta) = \sigma(0.2 + (0.3 \times 1 + (-2.2) \times 0 + 3.3 \times 1 + (-0.2) \times 2)) = 0.97$$

$$\sigma(x_C; \theta) = \sigma(0.2 + (0.3 \times 2 + (-2.2) \times 0 + 3.3 \times 0 + (-0.2) \times 1)) = 0.65$$

$$\sigma(x_D; \theta) = \sigma(0.2 + (0.3 \times 2 + (-2.2) \times 2 + 3.3 \times 0 + (-0.2) \times 0)) = 0.03$$

$$\sigma(x_E; \theta) = \sigma(0.2 + (0.3 \times 1 + (-2.2) \times 2 + 3.3 \times 1 + (-0.2) \times 7)) = 0.12$$

Step 2, now we can compute the parameter update for θ_1 . We are using the following formula:

$$\theta_1 = \theta_1 - \eta \sum_{i \in \{A, B, C, D, E\}} (\sigma(x_i; \theta) - y_i) x_{1i}$$

$$\theta_1 = 0.3 - 0.1 \sum_{i \in \{A,B,C,D,E\}} (\sigma(x_i; \theta) - y_i) x_{1i}$$

$$\begin{aligned} \theta_1 &= 0.3 - 0.1 [((\sigma(x_A; \theta) - y_A) \cdot x_{1A}) + ((\sigma(x_B; \theta) - y_B) \cdot x_{1B}) + ((\sigma(x_C; \theta) - y_C) \cdot x_{1C}) \\ &\quad + ((\sigma(x_D; \theta) - y_D) \cdot x_{1D}) + ((\sigma(x_E; \theta) - y_E) \cdot x_{1E})] \\ &= 0.3 - 0.1 [((0.94 - 1) \times 1) + ((0.97 - 1) \times 1) + ((0.65 - 1) \times 2) + ((0.03 - 0) \times 2) \\ &\quad + ((0.12 - 0) \times 1)] \\ &= 0.3 - 0.1((-0.06) + (-0.03) + (-0.70) + 0.06 + 0.12) = 0.3 - 0.1(-0.61) \\ &= 0.3 + 0.061 = 0.361 \end{aligned}$$

The new value for θ_1 is 3.061. Given the feedback from the current model mistakes (over on our training data), the *importance* of feature `Apple` has slightly increased. We can do the same thing for other parameters θ_2, θ_3 and θ_4 . We can continue the iterations until we find the “optimum” parameters.

NOTE: In the full Gradient Descent algorithm, we first compute the sigma value for all parameters (step 1 above), and then update **all parameters at once** (step 2 above).

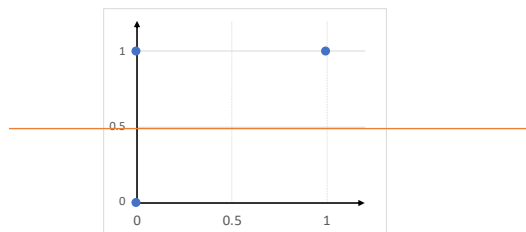
4. Consider the following training set:

(x_1, x_2)	y
(0,0)	0
(0,1)	1
(1,1)	1

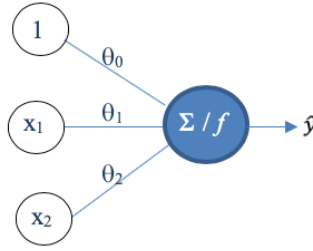
Consider the initial weight function as $\theta = \{\theta_0, \theta_1, \theta_2\} = \{0.2, -0.4, 0.1\}$ and the activation function of the perceptron as the step function of $f = \begin{cases} 1 & \text{if } \Sigma > 0 \\ 0 & \text{otherwise} \end{cases}$.

(i). Can the perceptron learn a perfect solution for this data set?

A perceptron can only learn perfect solutions for linearly separable problems, so you should ask yourself whether the data set is linearly separable. Indeed, it is. Imagine drawing the coordinates in a coordinate system, you will find that you can separate the positive example (0,0) from the negative ones ((0,1), (1,1)) with a straight line.



(ii). Draw the perceptron graph and calculate the accuracy of the perceptron on the training data before training?



To calculate the accuracy of the system we first need to calculate the output (prediction) of our perceptron and then compare it the actual labels of the class.

(x_1, x_2)	$\Sigma = \theta_0 + \theta_1 x_1 + \theta_2 x_2$	$\hat{y} = f(\Sigma)$	y
(0,0)	$0.2 - 0.4 \times 0 + 0.1 \times 0 = 0.2$	$f(0.2) = 1$	0
(0,1)	$0.2 - 0.4 \times 0 + 0.1 \times 1 = 0.3$	$f(0.3) = 1$	1
(1,1)	$0.2 - 0.4 \times 1 + 0.1 \times 1 = -0.1$	$f(-0.1) = 0$	1

As you can see one of the predictions (outputs) of our perceptron match the actual label and therefore the accuracy of our perceptron is $\frac{1}{3}$ at this stage.

- (iii). Using the perceptron *learning rule* and the learning rate of $\eta = 0.2$. Train the perceptron **for one epoch**. What are the weights after the training?

Remember the perceptron weights learning rule in iteration t and for train instance i is as follows:

$$\theta_j^{(t)} \leftarrow \theta_j^{(t-1)} + \eta(y^i - \hat{y}^{i,(t)}) x_j^i$$

For epoch 1 we will have:

(x_1, x_2)	$\Sigma = \theta_0 + \theta_1 x_1 + \theta_2 x_2$	$\hat{y} = f(\Sigma)$	y
(0,0)	$0.2 - 0.4 \times 0 + 0.1 \times 0 = 0.2$ Update θ : $\theta_0^{(1)} = \theta_0^{(0)} + \eta(y^1 - \hat{y}^{1,(1)}) x_0^1 = 0.2 + 0.2(0 - 1)1 = 0$ $\theta_1^{(1)} = \theta_1^{(0)} + \eta(y^1 - \hat{y}^{1,(1)}) x_1^1 = -0.4 + 0.2(0 - 1)0 = -0.4$ $\theta_2^{(1)} = \theta_2^{(0)} + \eta(y^1 - \hat{y}^{1,(1)}) x_2^1 = 0.1 + 0.2(0 - 1)0 = 0.1$	1	0
(0,1)	$0 - 0.4 \times 0 + 0.1 \times 1 = 0.1$ Correct prediction \rightarrow no update	1	1
(1,1)	$0 - 0.4 \times 1 + 0.1 \times 1 = -0.3$ Update θ : $\theta_0^{(1)} = \theta_0^{(0)} + \eta(y^3 - \hat{y}^{3,(1)}) x_0^3 = 0 + 0.2(1 - 0)1 = 0.2$ $\theta_1^{(1)} = \theta_1^{(0)} + \eta(y^3 - \hat{y}^{3,(1)}) x_1^3 = -0.4 + 0.2(1 - 0)1 = -0.2$ $\theta_2^{(1)} = \theta_2^{(0)} + \eta(y^3 - \hat{y}^{3,(1)}) x_2^3 = 0.1 + 0.2(1 - 0)1 = 0.3$	0	1

- (iv). What is the accuracy of the perceptron on the training data after training for one epoch? Did the accuracy improve?

With the new weights we get

- for instance (0,0) with $y=0$: $0.2 - 0.2 \times 0 + 0.3 \times 0 = 0.2$; $f(0.2) = 1$; **incorrect**
- for instance (0,1) with $y=1$: $0.2 - 0.2 \times 0 + 0.3 \times 1 = 0.5$; $f(0.5) = 1$; **correct**
- for instance (1,1) with $y=1$: $0.2 - 0.2 \times 1 + 0.3 \times 1 = 0.3$; $f(0.3) = 1$; **correct**

The accuracy of our perceptron is now $\frac{2}{3}$. So, the accuracy of the system has been improved 😊