School of Computing and Information Systems
The University of Melbourne
COMP90049 Introduction to Machine Learning (Semester 2, 2020)

Sample Solutions: Week 6

1. What is **Logistic Regression**?

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,…,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=1|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.

(i). How is **Logistic Regression** similar to **Naive Baye***s* and how is it different? In what circumstances would the former be preferable, and in what circumstances would the latter?

Similarity:
Both methods are classification methods, attempting to predict the most suitable class Y for a test instance X. Both methods compute P(Y|X) and predict the class with the highest probability. Thus, both methods are probabilistic machine learning models, basing their predictions around probabilities. Both classifiers have parameters, which are maximized using the Maximum Likelihood Principle. Both classifiers require input instances to be represented through a pre-defined set of features (and their appropriate values).

Difference:
Naïve Bayes is a *generative model*, which models P(x,y) = p(x|y) p(y), and maximizes the joint likelihood of the training data to find the best parameters. Naïve Bayes includes the *conditional feature independence* assumption, which ensures that the likelihood p(x|y) can be effectively estimated. Consider a visual metaphor: imagine we're trying to distinguish dog images from cat images. A generative model would have the goal of understanding what dogs look like and what cats look like.

On the other hand, Logistic Regression is a *discriminative* model, which models P(y|x) directly, and maximizes the conditional likelihood of the training data to find the best parameters. By *directly modelling p(y|x), there is no need to estimate p(x|y)*, so Logistic Regression does not require the feature independence assumption. A discriminative model is only trying to learn to *distinguish* between the classes (without learning much about them). So in our example, if all the dogs in the training data are wearing collars and the cats aren't. That one feature would neatly separates the classes and the model only predict the label for the test instances using that one feature (by assigning a very high weight to that feature). If you ask such a model what it knows about cats *all it can* say is that they don't wear collars!

Preferability:
Logistic Regression generally tends to outperform Naïve Bayes, however, in a situation where little training data is available Naïve Bayes can perform better than Logistic Regression. Naïve Bayes is conceptually simpler, and easier to implement, its parameters can be estimated in closed-form, whereas for Logistic Regression we have to adopt an iterative optimization method which can be time-consuming for large data.

(ii). What is "logistic"? What are we "regressing"?

We typically apply the logistic (sigmoid) function $\sigma = \frac{1}{1+e^{-z}}$ to the regression output z ($z = \vec{\theta}.\vec{X} = \theta_0 + \theta_1 x_1 + \cdots + \theta_F x_F$), where θ is the weight of the features. In other words, we are regressing the log odds of p(y=1|x).

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].

2. Bob tries to gather information about this year's apple harvest, and ran a search. He retrieved a number of articles, but found that a large portion of the retrieved articles are about the Apple laptops and computers -- and hence irrelevant to his search. He built the following data set of 5 training instances and 1 test instance. Develop a logistic regression classifier to predict label $\hat{y} = 1$ (fruit) and $\hat{y} = 0$ (computer).

ID	*apple*	*ibm*	*lemon*	*sun*	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					
T	1	2	1	5	0	COMPUTER

For the moment, we assume that we already have an estimate of the model parameters, i.e., the weights 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, and their meaning 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 modeller, 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, ..., x_F) directly as subject to parameter θ. Using the following:

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

Here, we have a binary output label *y = {0 (`computer`), 1 (`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

$$P(fruit|T) = P(\hat{y} = 1|T) = \sigma(\theta_0 + \theta_1 t_1 + \cdots + \theta_4 t_4)$$

$$= \sigma(0.2 + 0.3 \times 1 + (-2.2) \times 2 + 3.3 \times 1 + (-0.2) \times 5)$$

$$= \sigma(-1.6) = \frac{1}{1 + e^{-(-1.6)}} = 0.17$$

And consequently

$$P(computer|T) = 1 - P(fruit|T) = 1 - 0.17 = 0.83$$

Recall that we turn the logistic regression model into a classifier by predicting label y = 1 whenever p(y=1|x, θ) > 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). [OPTIONAL] Recall the conditional likelihood objective

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

We want to make sure that the Loss (the negative log likelihood) our model, is lower when its prediction the correct label for test instance T, than when it’s predicting a wrong label.

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 $\hat{y} = 0$:

$$\log \mathcal{L}(\theta) = -\sum_{i=1}^{n} 1 \log(\sigma(0.2 + 0.3 \times 1 + (-2.2) \times 2 + 3.3 \times 1 + (-0.2) \times 5)) + (1-1)\log(1 - \sigma(0.2 + 0.3 \times 1 + (-2.2) \times 2 + 3.3 \times 1 + (-0.2) \times 5))$$

$$= -\sum_{i=1}^{n} 1 \log(\sigma(-1.6)) + (1-1)\log(1 - \sigma(-1.6))$$

$$= -[1\, log(\sigma(-1.6)) + 0] = -log\, \frac{1}{1 + e^{-(-1.6)}} = -\log(0.17) = 1.77$$

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

$$\log \mathcal{L}(\theta) = -\sum_{i=1}^{n} 0 \log(\sigma(-1.6)) + (1-0)\log(1 - \sigma(-1.6))$$

$$= -\left[0 + 1\log\left(1 - \sigma(-1.6)\right)\right] = -log\,(1 - 0.17) = -\log(0.83) = 0.19$$

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.

3. For the model created in question 3, 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

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

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 $\eta = 0.1$.

$\theta_1$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 minimise 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:

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

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

$$\theta_1 = 0.3 - 0.1\,[((\sigma(x_A;\theta) - y_A).x_{1A}) + ((\sigma(x_B;\theta) - y_B).x_{1B}) + ((\sigma(x_C;\theta) - y_C).x_{1C})$$

$$+((\sigma(x_D;\theta) - y_D).x_{1D}) + ((\sigma(x_E;\theta) - y_E).x_{1E})]$$

$$= 0.3 - 0.1\,[((0.94 - 1) \times 1) + ((0.97 - 1) \times 1) + ((0.65 - 1) \times 2) + ((0.03 - 0) \times 2)$$

$$+((0.12 - 0) \times 1)]$$

$$= 0.3 - 0.1\left((-0.06) + (-0.03) + (-0.70) + 0.06 + 0.12\right) = 0.3 - 0.1\,(-0.61)$$
$$= 0.3 + 0.061 = 0.361$$

The new value for θ_1 is 0.361. 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. [OPTIONAL] What is the relation between "odds" and "probability"?

A probability is the **ratio of number of successes to the total possible outcomes**. For example in an experiment involving drawing a ball from a bag containing 8 balls (5 of them `red` and the rest `blue`), the probability of choosing a `red` ball is 5/8.

Odds on the other hand is the **ratio of the probability of success to the probability of failure**. Returning to our example, if we want to compute the odds of drawing a `red` ball, probability of *success* (draw `red`) over probability of *failure* (not draw `red`) is $\frac{\frac{5}{8}}{\frac{3}{8}} = \frac{5}{3} = 1.7$

Note that the odds considers two possible outcomes (success vs. failure) so that we can write the denominator *p(failure) = 1 - p(success).*

$$Odds = \frac{P(x)}{1 - P(x)}$$

So in our example the odds of choosing the 'red' can also calculated as follow:

$$Odds('red') = \frac{P('red')}{1 - P('red')} = \frac{\frac{5}{8}}{1 - \frac{5}{8}} = \frac{\frac{5}{8}}{\frac{3}{8}} = \frac{5}{3} = 1.7$$

5. [OPTIONAL] Why is a perceptron (which uses a **sigmoid** activation function) equivalent to logistic regression?

A perceptron has a weight associated with each input (attribute); the output is acquired by applying the activation function. This activation function can be the **step function** (as shown in the lectures) or any other (appropriate) function. If we use the **sigmoid** activation function ($\sigma(x) = f(x) = \frac{1}{1+e^{-x}}$) to the linear combination of inputs ($\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots$), which simplifies to $f(\theta^T x) = \sigma(\theta^T x)$— this is the same as the logistic regression function.

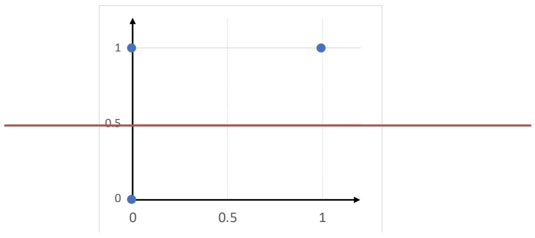
6. Consider the following training set:

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

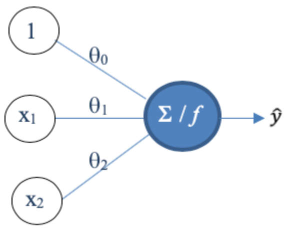
Consider the initial weight function as $\theta = \{\theta_0, \theta_1, \theta_2\}$ = {0.2, -0.4, 0.1} and the activation function of the perceptron as the step function of $f = \begin{cases} 1 & if\ \Sigma > 0 \\ 0 & otherwise \end{cases}$.

(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 x 0 + 0.1 x 0 = 0.2	*f*(0.2) = 1	0
(0,1)	0.2 – 0.4 x 0 + 0.1 x 1 = 0.3	*f*(0.3) = 1	1
(1,1)	0.2 – 0.4 x 1 + 0.1 x 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:

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

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 x 0 + 0.1 x 0 = 0.2	1	0
	Update θ:		
	$\theta_0^{(1)} = \theta_0^{(0)} + \eta\left(y^1 - \hat{y}^{1,(1)}\right) x_0^1 = 0.2 + 0.2\,(0-1)\,1 = 0$		
	$\theta_1^{(1)} = \theta_1^{(0)} + \eta\left(y^1 - \hat{y}^{1,(1)}\right) x_1^1 = -0.4 + 0.2\,(0-1)\,0 = -0.4$		
	$\theta_2^{(1)} = \theta_2^{(0)} + \eta\left(y^1 - \hat{y}^{1,(1)}\right) x_2^1 = 0.1 + 0.2\,(0-1)\,0 = 0.1$		
(0,1)	0 – 0.4 x 0 + 0.1 x 1 = 0.1	1	1
	Correct prediction → no update		
(1,1)	0 – 0.4 x 1 + 0.1 x 1 = – 0.3	0	1
	Update θ:		
	$\theta_0^{(1)} = \theta_0^{(0)} + \eta\left(y^3 - \hat{y}^{3,(1)}\right) x_0^3 = 0 + 0.2\,(1-0)\,1 = 0.2$		
	$\theta_1^{(1)} = \theta_1^{(0)} + \eta\left(y^3 - \hat{y}^{3,(1)}\right) x_1^3 = -0.4 + 0.2\,(1-0)\,1 = -0.2$		
	$\theta_2^{(1)} = \theta_2^{(0)} + \eta\left(y^3 - \hat{y}^{3,(1)}\right) x_2^3 = 0.1 + 0.2\,(1-0)\,1 = 0.3$		

(iv). [OPTIONAL] 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 x 0 + 0.3 x 0 = 0.2 ; f(0.2) = 1 ; **incorrect**
- for instance (0,1) with y=1 : 0.2 – 0.2 x 0 + 0.3 x 1 = 0.0 ; f(0.5) = 1 ; **correct**
- for instance (1,1) with y=1 : 0.2 – 0.2 x 1 + 0.3 x 1 = 0.1 ; f(0.3) = 1 ; **correct**

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