School of Computing and Information Systems
The University of Melbourne
COMP30027 Machine Learning (Semester 1, 2021)

Sample Solutions: Week 7

1. What is **gradient descent**? Why is it important?

   Gradient descent is an instance of iterative optimization algorithms: it finds the parameters corresponding to optimal points of a target function step-by-step, by starting out with some initial parameter value, and incrementally modifying this value in the `most promising way', i.e., in the way that leads to the largest improvement of the target function. This step-by-step procedure is important for optimization problems with no closed-form solution. This has numerous applications - most intuitively, in determining the regression weights which minimise an error function over some training data set.

2. What is Linear Regression? In what circumstances is it desirable, and in what circumstances it is undesirable?

   We attempt to build a linear model to predict the target values, by finding a weight for each attribute. The prediction is therefore $\sum_i w_i x_i$ (the point on the line). Remember that our line can have multiple dimensions (i.e., can be a hyperplane). The fact is many (but not all) relationships can be approximately described by such a model.

   We build a linear regression model by learning (tunning) the weights using Gradient Descent. In other words, we use gradient decent to minimise the error of our model predictions with respect to an error function.

   MSE is one popular error function. Using MSE, we attempt to minimise the squared differences between our predicted values and the actual target values from the training data. The main benefit of MSE is that it is convex — so that we know that there is a unique best solution.

3. Recall that the update rule for Gradient Descent with respect to Mean Squared Error (MSE) is as follows:

$$\beta_k^{j+1} = \beta_k^j + \frac{2\alpha}{N}\sum_{i=1}^{N} x_{ik}(y_i - \hat{y}_i^j)$$

   Build a Linear Regression model, using the following instances:

x	y
1	1
2	2
2	3

   Recall that gradient descent:

   - iteratively improves our estimates of the prediction line (according to parameters β)
   - is based on the partial derivatives (which together define the gradient) of the error function (in this case, MSE)
   - tries to find a weight (β_k) for each attribute (k ∈ [1..D]), including a dummy attribute numbered 0 (known as the "bias")

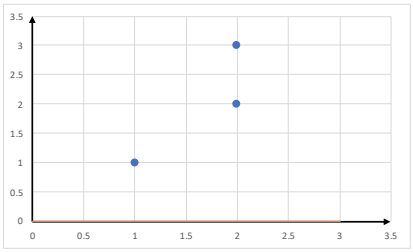
   We need to specify an initial estimate for the weights — here, I will use $\hat{y} = 0 + 0x$. Recall that because MSE is **convex**, the choice of initial estimate shouldn't affect our overall answer.

   We also need to choose the learning rate α; here we will say $\frac{\alpha}{N}$ = 0.05

Skipping the workings, our update rule for each β_k is based on the error of our prediction $\hat{y}$ for each instance $(y_i - \hat{y}_i)$:

$$\beta_k^{(1)} = \beta_k + \frac{2\alpha}{N}\sum_i x_{ik}(y_i - \hat{y}_i)$$

Here, we have 3 (N) instances, and we are asked to begin with the line $\hat{y} = 0 + 0x$ (so β = <0,0>)



A good place to begin is to calculate the errors for our instances (note that these would be squared, if we were calculating MSE):

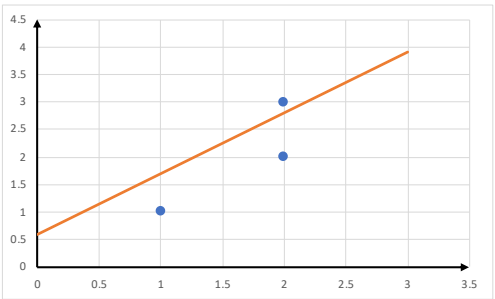
- for the first point, we predict $\hat{y}_1$ = 0(1) + 0 = 0, but we have y_1 = 1 (so the error is +1)
- for the second point, we predict $\hat{y}_2$ = 0(2) + 0 = 0, but we have y_1 = 2 (so the error is +2)
- for the third point, we predict $\hat{y}_3$ = 0(2) + 0 = 0, but we have y_1 = 3 (so the error is +3)

Our first update will look as follows:

$$\beta_0^{(1)} = \beta_0 + \frac{2\alpha}{N}\sum_i x_{i0}(y_i - \hat{y}_i) = 0 + 2(0.05)[(1)(1-0) + (1)(2-0) + (1)(3-0)] = 0.6$$

$$\beta_1^{(1)} = \beta_1 + \frac{2\alpha}{N}\sum_i x_{i1}(y_i - \hat{y}_i) = 0 + 2(0.05)[(1)(1-0) + (2)(2-0) + (2)(3-0)] = 1.1$$

So, our model has changed from $\hat{y} = 0 + 0x$ to line $\hat{y} = 1.1x + 0.6$.



We can see that it is an improvement, because the (squared) errors have reduced:

- for the first point, we predict $\hat{y}_1 = 1.1 \times 1 + 0.6$ = 1.7, but we have y_1 = 1 (so the error is −0.7)
- for the second point, we predict $\hat{y}_2 = 1.1 \times 2 + 0.6$ = 2.8, but we have y_2 = 2 (so the error is −0.8)
- for the third point, we predict $\hat{y}_3 = 1.1 \times 2 + 0.6$ = 2.8, but we have y_3 = 3 (so the error is +0.2)

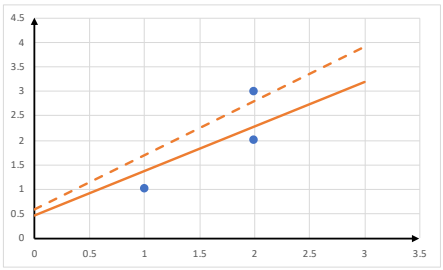
Note that the sign matters in the gradient descent updates, even if it doesn't when we calculate MSE. (This is an easy mistake to make!)

The second update proceeds the same way:

$$\beta_0^{(2)} = \beta_0^{(1)} + \frac{2\alpha}{N}\sum_i x_{i0}(y_i - \hat{y}_i) = 0.6 + 2(0.05)[(1)(1-1.7) + (1)(2-2.8) + (1)(3-2.8)] = 0.47$$

$$\beta_1^{(2)} = \beta_1^{(1)} + \frac{2\alpha}{N}\sum_i x_{i1}(y_i - \hat{y}_i) = 1.1 + 2(0.05)[(1)(1-1.7) + (2)(2-2.8) + (2)(3-2.8)] = 0.91$$

Our new line is $\hat{y} = 0.91x + 0.47$.



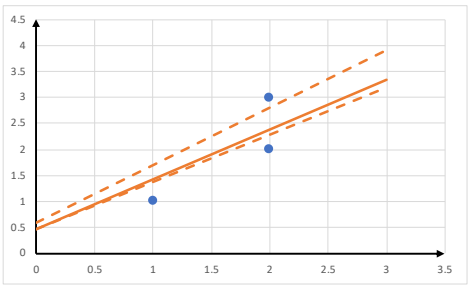
We can see that this new line is again improved the (squared) errors:

- for the first point, we predict $\hat{y}_1 = 0.91 \times 1 + 0.47 = 1.38$, but we have $y_1 = 1$ (so the error is −0.38)
- for the second point, we predict $\hat{y}_2 = 0.91 \times 2 + 0.47 = 2.29$, but we have $y_2 = 2$ (so the error is −0.29)
- for the third point, we predict $\hat{y}_3 = 0.91 \times 2 + 0.47 = 2.29$, but we have $y_3 = 3$ (so the error is +0.71)

Let's do a few more updates, so we can see what's happening with our weights:

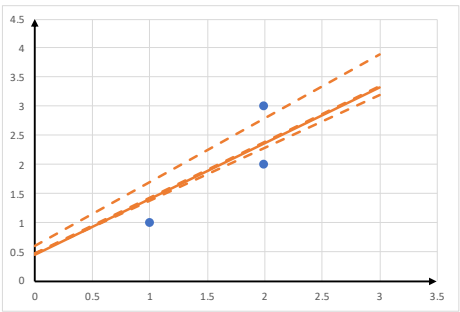
$$\beta_0^{(3)} = 0.47 + 2(0.05)[(1)(1 - 1.38) + (1)(2 - 2.29) + (1)(3 - 2.29)] = 0.474$$

$$\beta_1^{(3)} = 0.91 + 2(0.05)[(1)(1 - 1.38) + (2)(2 - 2.29) + (2)(3 - 2.29)] = 0.956$$



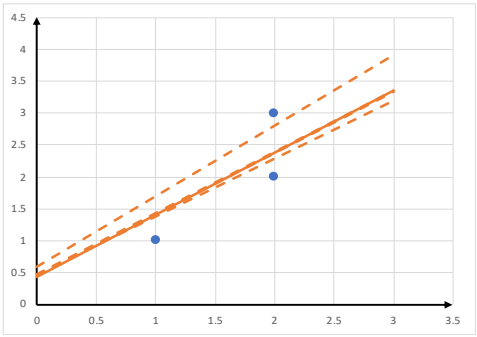
$$\beta_0^{(4)} = 0.474 + 2(0.05)[(1)(1 - 1.43) + (1)(2 - 2.386) + (1)(3 - 2.389)] \approx 0.454$$

$$\beta_1^{(4)} = 0.956 + 2(0.05)[(1)(1 - 1.43) + (2)(2 - 2.386) + (2)(3 - 2.389)] \approx 0.959$$



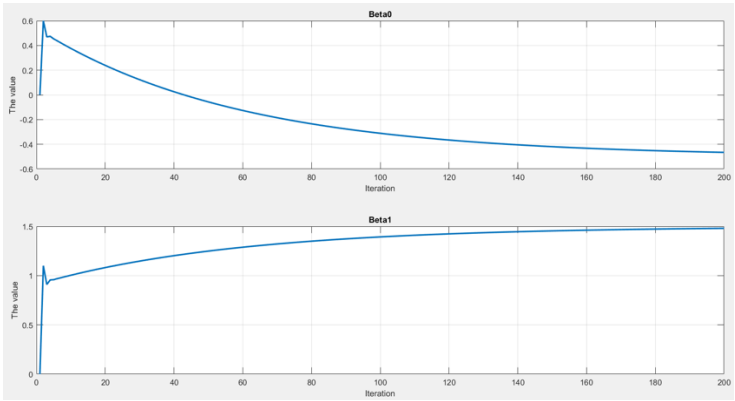
$$\beta_0^{(5)} = 0.454 + 2(0.05)[(1)(1 - 1.413) + (1)(2 - 2.372) + (1)(3 - 2.372)] \approx 0.438$$

$$\beta_1^{(5)} = 0.959 + 2(0.05)[(1)(1 - 1.413) + (2)(2 - 2.372) + (2)(3 - 2.372)] \approx 0.969$$



After many iterations (almost 200 iterations) our line converges to $\hat{y} = 1\,x - 0.5$. You can see that our initial choice was pretty poor, so we made some drastic changes to try to improve it; β_0 initially moved in the wrong direction, but it is (slowly) being corrected now.

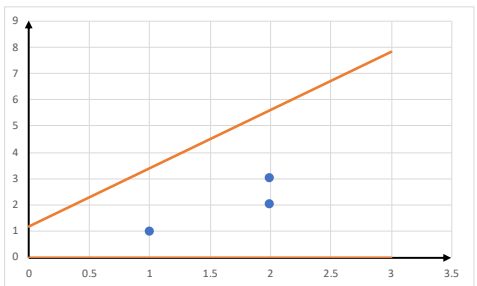
You can see the pattern of changes for β_0 and β_1 in the following diagram.



This looks pretty slow; **what if we use a larger learning rate ($\frac{\alpha}{N}$ = 0.1)**?

$$\beta_0' = \beta_0 + \frac{2\alpha}{N}\sum_i x_{i0}(y_i - \hat{y}_i) = 0 + 2(0.1)[(1)(1-0) + (1)(2-0) + (1)(3-0)] = 1.2$$

$$\beta_1' = \beta_1 + \frac{2\alpha}{N}\sum_i x_{i1}(y_i - \hat{y}_i) = 0 + 2(0.1)[(1)(1-0) + (2)(2-0) + (2)(3-0)] = 2.2$$



Is this line ($\hat{y} = 2.2\,x + 1.2$) an improvement?

- for the first point, we predict $\hat{y}_1 = 2.2 \times 1 + 1.2 = 3.4$, but we have $y_1 = 1$ (so the error is −2.4)
- for the second point, we predict $\hat{y}_2 = 2.2 \times 2 + 1.2 = 5.6$, but we have $y_2 = 2$ (so the error is −3.6)
- for the third point, we predict $\hat{y}_3 = 2.2 \times 2 + 1.3 = 5.6$, but we have $y_3 = 3$ (so the error is -2.6)

If we square these values (25.5), we will indeed find that this is actually worse than our initial error (14). We have increased our error rate too much, and we won't actually achieve convergence.

The good news is that — since we are calculating the errors for all of the instances anyway, it isn't too much extra work to "sanity check" that the MSE is actually decreasing.

- If we're in the first few steps and MSE is increasing, then we can start again, with a different initial guess and/or learning rate.
- If we're quite a few steps along, and MSE is increasing, then we can call this "convergence", or — if we need more accuracy — take our current estimate of β and reduce the learning rate.

4. 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 $[x_1, x_2,\ldots,x_n]$. The output y can be 1 or 0 (i.e., winning / not winning).

The model uses linear regression to calculate the log odds of 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.

We typically apply the logistic (sigmoid) function $\sigma = \frac{1}{1+e^{-z}}$ to the regression output z ($z = \vec{\beta}.\vec{X} = \beta_0 + \beta_1 x_1 + \cdots + \beta_F x_F$), where β 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].

5. [OPTIONAL] What is the relation between "odds" and "probability"?

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 consider 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:

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

6. Build a logistic regression classifier, which uses the counts of selected words in the news articles to predict the class of the news article (fruit vs. computer). Assume the weights of the 4 features (and the bias β_0) is $\widehat{\beta} = [\beta_0, \beta_1, \beta_2, \beta_3, \beta_4] = [0.2, 0.3, -2.2, 3.3, -0.2]$.

ID	*apple*	*ibm*	*lemon*	*sun*	CLASS
TRAINING INSTANCE					
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

a) 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 β. Using the following:

$$P(y = 1|x_1, x_2, \dots, x_F) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_F x_F)}} = \sigma(\beta_0 + \beta_1 x_1 + \cdots + \beta_F x_F)$$

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

Weights $\beta_1, \beta_2, \beta_3, \beta_4$ denote the respective importance of the features 1 to 4 for predicting the class `fruit` (1). For example, β_2 (-2.2) indicates how important feature 2 (`ibm`) is for predicting $\hat{y}$ = 1 (fruit). β_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 β_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`.

b) Predict the test label.

Using the logistic regression formula above, we find

$$P(fruit|T) = P(\hat{y} = 1|T) = \sigma(\beta_0 + \beta_1 t_1 + \cdots + \beta_4 t_4)$$

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

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

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 $p(y=1|x, \beta) > 0.5$, 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. ☺

7. For the model created in question 6, compute a single gradient descent update for parameter β_1 given the training instances given above. Recall that for each feature j, we compute its weight update as

$$\beta_j \leftarrow \beta_j + \alpha \frac{\partial \mathcal{L}(\beta)}{\partial \beta_j}$$

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

θ_1 is the model parameter (respective importance) of the features 1 (apple) in our model. Using Gradient Ascent method over iterations, we want to find the best parameters of the model (the parameters that maximise the likelihood of our model). Gradient Ascent has the same principle as Gradient Descent, but it goes up-hill and maximises the target function.

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

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

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

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

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

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

Step 2, now we can compute the parameter update for β_1. We are using the following formula:

$$\beta_j \leftarrow \beta_j + \alpha \frac{\partial \mathcal{L}(\beta)}{\partial \beta_j}$$

Using gradient decent concept, the derivative of our loss function can be calculated as $\sum_i (y_i - \sigma(x_i; \beta))\, x_{1i}$ so our update formula for parameter update.

$$\beta_1 = \beta_1 + \alpha \sum_{i \in \{A,B,C,D,E\}} (y_i - \sigma(x_i; \beta))\, x_{1i}$$

$$\beta_1 = 0.3 + 0.1 \sum_{i \in \{A,B,C,D,E\}} (y_i - \sigma(x_i; \beta))\, x_{1i}$$

$$\beta_1 = 0.3 + 0.1\,[((y_A - \sigma(x_A; \beta)).x_{1A}) + ((y_B - \sigma(x_B; \beta)).x_{1B}) + ((y_C - \sigma(x_C; \beta)).x_{1C}) + ((y_D - \sigma(x_D; \beta)).x_{1D}) + ((y_E - \sigma(x_E; \beta)).x_{1E})]$$

$$= 0.3 + 0.1\,[((1 - 0.94) \times 1) + ((1 - 0.97) \times 1) + ((1 - 0.65) \times 2) + ((0 - 0.03) \times 2) + ((0 - 0.12) \times 1)]$$

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

The new value for β_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 β_2, β_3 and β_4. We can continue the iterations until we find the "optimum" parameters.

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