School of Computing and Information Systems
The University of Melbourne
COMP30027 MACHINE LEARNING (Semester 1, 2019)

Tutorial sample solutions: Week 7

1. What is the **Gradient Descent** method, and why is it important?

   - Gradient Descent is a mechanism for finding the minimum of a (convex) multivariate function, where we can find its partial derivatives.
   - This has numerous applications — most intuitively, in determining the regression weights which minimise an error function over some training data set.

2. What is **Regression**? How is it similar to **Classification**, and how is it different?

   - Our target attribute (class) is nominal in classification, but numeric (continuous) in Regression.
   - Consequently, we can't exhaustively assess the likelihood of each class.

   (a) What is **Linear Regression**? In what circumstances is it desirable, and it what circumstances is it undesirable?

   - We attempt to build a linear model to predict the target values, by finding a weight for each attribute. The prediction is therefore $\sum_i w_i a_i$
   - Many (but not all) relationships can be approximately described by such a model.

   (b) How do we build a (linear) regression model? What is **RSS** and what advantages does it have over (some) alternatives?

   - By learning the weights using Gradient Descent, with respect to an error function.
   - This assumes that our error function is convex — so that there is a unique solution. RSS is one popular choice for this: where we attempt to minimise the sum of the squared differences between our predicted values and the actual target values from the training data.

3. Recall that the update rule for Gradient Descent with respect to RSS is as follows:

$$\beta_k^{i+1} \quad := \quad \beta_k^i + 2\alpha \sum_{j=1}^{N} x_{jk}(y_j - \hat{y_j^i})$$

   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, RSS)
     - tries to find a weight (β_k) for each attribute ($k \in [1..D]$), 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 RSS 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 $\alpha = 0.05$
- Skipping the workings, our update rule for each β_k is based on the error of our prediction $\hat{y}$ for each instance $(y_i - \hat{y}_i)$:

$$\beta'_k = \beta_k + 2\alpha \sum_i x_{ik}(y_i - \hat{y}_i)$$

- Here, we have 3 (N) instances, and we are asked to begin with the line $\hat{y} = 0x + 0$ (so $\beta = \langle 0, 0 \rangle$)
- A good place to begin is to calculate the errors for our instances (note that these would be squared, if we were calculating RSS):
  - 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:

$$\begin{aligned}
\beta'_0 &= \beta_0 + 2\alpha \sum_i x_{i0}(y_i - \hat{y}_i) \\
\beta'_0 &= 0 + 2(0.05)[(1)(1-0) + (1)(2-0) + (1)(3-0)] = 0.6 \\
\beta'_1 &= \beta_1 + 2\alpha \sum_i x_{i1}(y_i - \hat{y}_i) \\
\beta'_1 &= 0 + 2(0.05)[(1)(1-0) + (2)(2-0) + (2)(3-0)] = 1.1
\end{aligned}$$

- We can see now that this line ($\hat{y} = 1.1x + 0.6$) is an improvement, because the (squared) errors have reduced:
  - for the first point, we predict $\hat{y_1} = 1.1(1) + 0.6 = 1.7$, but we have $y_1 = 1$ (so the error is -0.7)
  - for the second point, we predict $\hat{y_2} = 1.1(2) + 0.6 = 2.8$, but we have $y_1 = 2$ (so the error is -0.8)
  - for the third point, we predict $\hat{y_3} = 1.1(2) + 0.6 = 2.8$, but we have $y_1 = 3$ (so the error is $+0.2$)
- Note that the sign matters in the gradient descent updates, even if it doesn't when we calculate RSS. (This is an easy mistake to make!)
- The second update proceeds the same way:

$$\begin{aligned}
\beta''_0 &= \beta'_0 + 2\alpha \sum_i x_{ik}(y_i - \hat{y}_i) \\
\beta''_0 &= 0.6 + 2(0.05)[(1)(1-1.7) + (1)(2-2.8) + (1)(3-2.8)] = 0.47 \\
\beta''_1 &= \beta'_1 + 2\alpha \sum_i x_{ik}(y_i - \hat{y}_i) \\
\beta''_1 &= 1.1 + 2(0.05)[(1)(1-1.7) + (2)(2-2.8) + (2)(3-2.8)] = 0.91
\end{aligned}$$

- Our new line ($\hat{y} = 0.91x + 0.47$) has again improved the (squared) errors:
  - for the first point, we predict $\hat{y_1} = 0.91(1) + 0.47 = 1.38$, but we have $y_1 = 1$ (so the error is -0.38)
  - for the second point, we predict $\hat{y_2} = 0.91(2) + 0.47 = 2.29$, but we have $y_1 = 2$ (so the error is -0.29)
  - for the third point, we predict $\hat{y_3} = 0.91(2) + 0.47 = 2.29$, but we have $y_1 = 3$ (so the error is $+0.71$)

- Let's do a few more updates, so we can see what's happening with β_0:

$$\begin{aligned}\beta_0^{(3)} &= 0.47 + 2(0.05)[(1)(1-1.38) + (1)(2-2.29) + (1)(3-2.29)] = 0.474\\ \beta_1^{(3)} &= 0.91 + 2(0.05)[(1)(1-1.38) + (2)(2-2.29) + (2)(3-2.29)] = 0.956\end{aligned}$$

$$\begin{aligned}\beta_0^{(4)} &= 0.474 + 2(0.05)[(1)(1-1.43) + (1)(2-2.386) + (1)(3-2.386)] \approx 0.454\\ \beta_1^{(4)} &= 0.956 + 2(0.05)[(1)(1-1.43) + (2)(2-2.386) + (2)(3-2.386)] \approx 0.959\end{aligned}$$

$$\begin{aligned}\beta_0^{(5)} &= 0.454 + 2(0.05)[(1)(1-1.413) + (1)(2-2.372) + (1)(3-2.372)] \approx 0.438\\ \beta_1^{(5)} &= 0.959 + 2(0.05)[(1)(1-1.413) + (2)(2-2.372) + (2)(3-2.372)] \approx 0.969\end{aligned}$$

- We can observe the optimal line by inspection $\hat{y} = 1.5x - 0.5$. 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.
- This looks pretty slow; what if we use a larger learning rate ($\alpha = 0.1$)?
- Our first update will look as follows:

$$\begin{aligned}\beta_0' &= \beta_0 + 2\alpha \sum_i x_{ik}(y_i - \hat{y}_i)\\ \beta_0' &= 0 + 2(0.1)[(1)(1-0) + (1)(2-0) + (1)(3-0)] = 1.2\\ \beta_1' &= \beta_1 + \frac{2\alpha}{N} \sum_i x_{ik}(y_i - \hat{y}_i)\\ \beta_1' &= 0 + 2(0.1)[(1)(1-0) + (2)(2-0) + (2)(3-0)] = 2.2\end{aligned}$$

- Is this line ($\hat{y} = 2.2x + 1.2$) an improvement?
  - for the first point, we predict $\hat{y_1} = 2.2(1) + 1.2 = 3.4$, but we have $y_1 = 1$ (so the error is -2.4)
  - for the second point, we predict $\hat{y_2} = 2.2(2) + 1.2 = 5.6$, but we have $y_1 = 2$ (so the error is -3.6)
  - for the third point, we predict $\hat{y_3} = 2.2(2) + 1.2 = 5.6$, but we have $y_1 = 3$ (so the error is -2.6)
- If we square these values (25.5), we would indeed find that this is actually worse than our initial guess (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 RSS is actually decreasing.
  - If we're in the first few steps and RSS 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 RSS 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**?

- We build a (linear) regression model, where the target is (close to) 1 for instances of the positive class, and (close to) 0 for instance of the negative class. (Suitably re-interpretted for multi-class problems.)

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

- Effectively, both methods are attempting to find the class c for a test instance T, by maximising $P(c|T)$.
- In Naive Bayes, we make some simplifying assumptions, most notably, that the attributes are conditionally independent of the class labels.
- In Logistic Regression, we attempt to model this directly, without the simplifying assumptions. This is possible because we don't attempt to generate the class probabilities; we only attempt to discriminate amongst the various classes.

(b) What is "logistic"? What are we "regressing"?

- We typically apply the logistic function $\frac{1}{1+e^{-m}}$ to the regression output ($\beta \cdot x$), which has an easy–to–calculate derivative, and has a range of $[0, 1]$.

(c) How do we train a Logistic Regression model? In particular, what is the significance of the following:

$$\text{argmax}_\beta \sum_{i=1}^{n} y_i \log h_\beta(x_i) + (1 - y_i) \log(1 - h_\beta(x_i))$$

- This is a confusing way of stating that we want the output of the linear regression to be (very) positive when the target class is 1, and (very) negative when the target class is 0.
- We want to choose a set of parameters β which maximises this objective function; we use Gradient Ascent to find them.