UNIVERSITY OF OXFORD

Machine Learning
Michaelmas Term 2021
Week 3

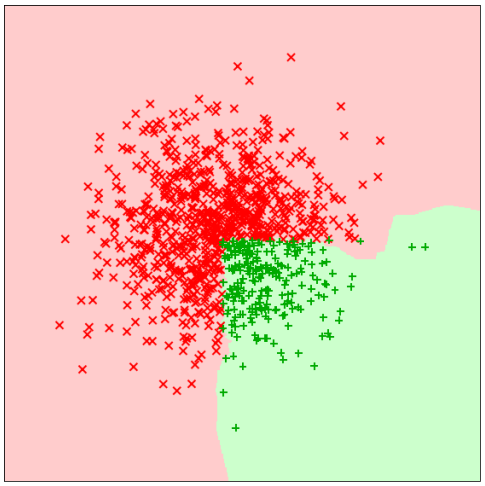
Problem Sheet 1 Solutions

1 Nearest Neighbour Classification

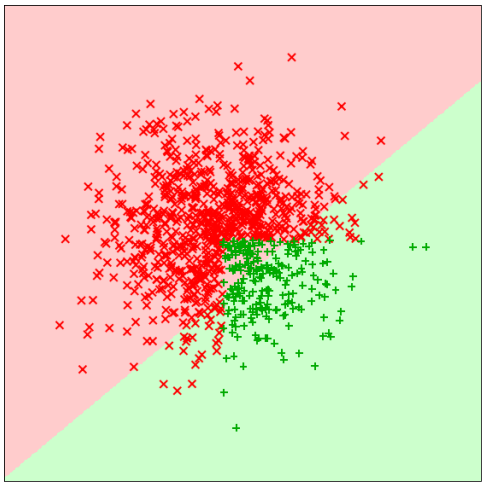
In the lectures, we studied the perceptron, a linear classifier of the form $y = \text{sign}(\mathbf{w} \cdot \mathbf{x} + w_0)$, where $\text{sign}(z) = 1$ if $z \geq 0$ and $\text{sign}(z) = 0$ otherwise. The parameters to be learnt are $\mathbf{w}$ and w_0. The "Nearest neighbour classifier" (NN) is a different approach to learning from data. Suppose we are given N points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ where $y_i \in \{0, 1\}$; for a parameter k and given a new point $\mathbf{x}^*$, the k-NN approach does the following: find $\mathbf{x}_{j_1}, \ldots \mathbf{x}_{j_k}$ the k-closest points to $\mathbf{x}^*$, then output $\widehat{y}^*$ as the majority label from the set $\{y_{j_1}, \ldots, y_{j_k}\}$, *i.e.,* the most commonly occurring label among the k-nearest neighbours.

1. What advantage does the k-NN approach offer over a linear classifier like the perceptron?

   Solution: The k-NN approach allows us to represent very general functions. The only assumption is that the functions are somewhat smooth so that nearby points take values that are close to each other. On the other hand, linear classification (perceptron) makes a very strong assumption about the relationship between output and input, namely that the positively labelled points and negatively labelled points are linearly separable. If this assumption is not satisfied by the actual data, there will be significant errors. The plots below show the decision boundaries for the k-NN classifier with $k = 15$ (Fig. (a)) and a linear classifier (Fig. (b)) on the same data. Notice how the k-NN decision boundary adjusts to the actual labels of the data, whereas any linear classifier makes errors as the data is not separable by a hyperplane (in this case by a line in the plain).



(a) k-NN



(b) Linear Classifier

2. How many parameters does the nearest neighbour model have? How much memory do you need to store the model? What is the computational cost of predicting the label $\widehat{y}^*$?

   Solution: Naïvely one might think that there is only one parameter in this approach, the number k. However, in reality one has to store the entire training dataset to be able to make any predictions. Thus, this should really be considered as a non-parametric model where the model is a function of the entire training dataset. If the training data consists of N points, we need to store N vectors. Unfortunately the computational cost can be linear in N to make a prediction on a new point. However, with additional assumptions and if we settle for approximate nearest neighbours there are techniques to speed this up somewhat.

3. In this part, we'll look at the setting where the vectors $\mathbf{x}$ are points on the boolean hypercube, *i.e.,* $\mathbf{x} \in \{0,1\}^D$. Fix $\mathbf{x}^* = (0, 0, \ldots, 0)$ to be the origin and imagine that data consists of points drawn uniformly at random from the boolean hypercube. What is the distribution of the Hamming distance of data points from $\mathbf{x}^*$? What happens as $D \to \infty$? (*Hint*: Use the central limit theorem.)

   Solution: We can assume that each co-ordinate of the input point is drawn independently. Thus, each co-ordinate makes a contribution of 1 to the Hamming distance with probability $1/2$ and 0 otherwise. Thus, the distance is distributed as Binomial$(D, 1/2)$. In the limit that $D \to \infty$, this approaches a Gaussian distribution with mean $D/2$ and variance $D/4$ using the CLT.

4. Let us now fix some numbers. Suppose the dimension of the data $D = 10{,}000$; let $\mathbf{x}^* = (0, 0, \ldots, 0)$ and suppose we generated $N = 10{,}000$ data points. What do you expect the distance of $\mathbf{x}^*$ from the nearest data-point to be? the furthest? How large does N need to be to get points that are reasonably close to $\mathbf{x}^*$, say within Hamming distance 50?

   Solution: The fact that the distance of randomly drawn points is distributed like a Gaussian (or for that matter binomial) means that the closest points to $\mathbf{x}^*$ are about as far from $\mathbf{x}^*$ as the furthest points. Note that the standard deviation is $O(\sqrt{D})$ which is much smaller than the mean distance which is $D/2$. Thus to get points that are meaningfully close to any particular point, one needs to start with a set of points that is exponentially large in the dimension. Thus, the nearest-neighbour approach is not effective in high-dimensions even in the age of big-data! When the dimension is small, these approaches can be extremely effective and much more flexible than fixed parametric models.

Remark: You do not have to write precise numbers or even mathematical expressions for the answers to part 4 above. Make sure you understand the behaviour qualitatively. The phenomenon explored in the last two parts of the question is referred to as the *curse of dimensionality*.

2 Logical Gates Using Perceptrons

Recall that a perceptron with input features $x_1, \ldots, x_D$, weights $w_1, \ldots, w_D$ and bias w_0 outputs the value:

$$y = \begin{cases} 1 & \text{if } w_0 + \sum_{i=1}^{D} w_i x_i \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

1. Suppose there are at most two inputs and the inputs always take binary values, *i.e.,* $x_i \in \{0, 1\}$. Show how to construct AND, OR and NOT gates by suitably adjusting weights.

   Solution: Assuming that $\text{sign}(z) = 1$ for $z \geq 0$ and 0 otherwise, we have

   $$\begin{aligned} \text{AND}(x_1, x_2) &= \text{sign}(x_1 + x_2 - 2) \\ \text{OR}(x_1, x_2) &= \text{sign}(x_1 + x_2 - 1) \\ \text{NOT}(x_1) &= \text{sign}(-x_1) \end{aligned}$$

2. The constructions for AND and OR gates required only the bias term w_0 to be negative, all other weights were positive. Can you achieve a similar construction for the NOT gate? Why?

   Solution: No, because perceptron output is of the form $\text{sign}(w_0 + w_1 x_1)$ which is increasing in x_1 if $w_1 > 0$.

3. Can you construct an XOR (exclusive or) gate? If not, give reasons.

   Solution: No, it is easiest to see geometrically. The points labelled $y = 1$ by a perceptron can be separated from those labelled by $y = 0$ by a line in the plane (for two dimensional inputs).

   In the case of the XOR function, the points $(0, 1)$ and $(1, 0)$ have label $y = 1$ and the points $(0, 0)$ and $(1, 1)$ have label $y = 0$. It is easy to see that no line can separate the four points correctly.

4. Often, instead of using a hard threshold we would like to use a continuous approximation. Recall the hyperbolic tangent function $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. We consider another type of *artificial neuron* whose output is defined as

   $$y = \tanh\left(w_0 + \sum_{i=1}^{D} w_i x_i\right). \tag{2.2}$$

   Suppose you treat outputs above 0.99 as true and those below -0.99 as false. Show that similar constructions to the ones you had earlier can still be used to construct logic gates.

   Solution: We need to be sure that the value of the linear function (inside the sign) is

either strictly positive or negative (not zero) and scaled by a large enough constant so that the values of tanh are saturated close to 1 or -1.

$$\begin{aligned}\text{AND}(x_1, x_2) &= \text{sign}(200x_1 + 200x_2 - 300)\\ \text{OR}(x_1, x_2) &= \text{sign}(200x_1 + 200x_2 - 100)\\ \text{NOT}(x_1, x_2) &= \text{sign}(-200x_1 + 100)\end{aligned}$$

3 Share Price Prediction using Linear Regression

Note: This example is for illustrative purposes to help you understand linear regression. It is not recommended that you use this for actual share price prediction.

Suppose that $x_0, x_1, \ldots, x_t, x_{t+1}, \ldots$, denote the (daily) share prices of a particular stock over time. Answer the following questions (you should add a bias term as necessary):

1. Write a linear model to predict x_{t+1} using the share prices on the two preceeding days, *i.e.,* x_t and x_{t-1} .

   Solution:
   $$x_{t+1} = w_0 + w_1 x_t + w_2 x_{t-1} + \epsilon$$
   where ϵ above denotes the noise term.

2. The more useful quantity to predict is $\Delta_{t+1} := x_{t+1} - x_t$, the change in share value. Write a linear model to predict Δ_{t+1} using x_t and x_{t-1}.

   Solution:
   $$\Delta_{t+1} = w_0 + w_1 x_t + w_2 x_{t-1} + \epsilon$$
   where ϵ above denotes the noise term.

3. Write a linear model to predict Δ_{t+1} using Δ_t.

   Solution:
   $$\Delta_{t+1} = w_0 + w_1 \Delta_t + \epsilon$$
   where ϵ above denotes the noise term.

4. Write a linear model to predict Δ_{t+1} using Δ_t and x_t.

   Solution:
   $$\Delta_{t+1} = w_0 + w_1 x_t + w_2 \Delta_t + \epsilon$$
   where ϵ above denotes the noise term.

5. Which of the above four models, if any, are equivalent? Justify your answer briefly.

   Solution: The models described in Parts (1), (2) & (4) are identical. When the models in (2) & (4) are expanded out, it can be easily checked that x_{t+1} can be expressed as an

arbitrary affine function of the quantities x_t and x_{t-1}, exactly as in the case of Part (1). The model in Part (3) is more restrictive; there are probably several ways to see this, but the easiest is that it only involves two "free" variables as opposed to three "free" variables in the case of models (1), (2) and (4).

6. Given that the only observations you make is the sequence $(x_0, x_1, \ldots, x_t, x_{t+1}, \ldots, x_T)$ for some T, explain how you would train the model in Part (4) above.

   Solution: We simply divide the sequence into smaller sequences of length 3 to obtain the training data $\langle((x_1 - x_0, x_1), y_1 = x_2 - x_1), ((x_2 - x_1, x_2), y_2 = x_3 - x_2), \ldots, ((x_{T-1} - x_{T-2}, x_{T-1}), y_{T-1} = x_T - x_{T-1})\rangle$. Then the linear model can be trained in exactly the same way as we did in the lecture.