Machine Learning
Hilary Term 2016
Week 3

Problem Sheet 1

1 Nearest Neighbour Classification

In class we studied linear regression, where we model the output $y = \mathbf{w} \cdot \mathbf{x} + \text{noise}$, where $\mathbf{w}$ is the parameter vector to be learnt and $\mathbf{x}$ is the input. Nearest neighbour (NN) is a different approach to learning from data. Suppose we are given m points $(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_m, y_m)$; 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 compute

$$\hat{y}^* = \sum_{l=1}^{k} \alpha_{j_l} y_{j_l},$$

where α_{j_l} are weights inversely proportional to the distance between $\mathbf{x}$ and $\mathbf{x}_{j_l}$.

1. What advantage does the k-NN approach offer over linear regression?
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 producing $\hat{y}^*$?
3. In this part, we'll look at the setting where the vectors $\mathbf{x}$ are points on the boolean hypercube, *i.e.,* $\mathbf{x} \in \{0,1\}^n$. 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 $n \to \infty$? (*Hint*: Use the central limit theorem.)
4. Let us now fix some numbers. Suppose the dimension of the data $n = 10,000$; let $\mathbf{x}^* = (0, 0, \ldots, 0)$ and suppose we generated $m = 10,000$ data points. What do you expect the distance of $\mathbf{x}^*$ from the nearest data-point to be? the furthest? How large does m need to be to get points that are reasonably close to $\mathbf{x}^*$, say within Hamming distance 50?

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 Normalization constant for a 1D Gaussian

The normalization constant for a zero-mean Gaussian is given by

$$Z = \int_{-\infty}^{\infty} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx.$$

To compute this, consider its square

$$Z^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) dxdy.$$

Let us change variables from cartesian (x, y) to polar (r, θ) using $x = r\cos\theta$ and $y = r\sin\theta$. Since $dxdy = rdrd\theta$ (recall that r is the determinant of the Jacobian in 2D) and $cos^2\theta + \sin^2\theta = 1$, we have

$$Z^2 = \int_0^{2\pi}\int_0^{\infty} r\exp\left(-\frac{r^2}{2\sigma^2}\right) drd\theta$$

Evaluate this integral and hence show $Z = \sqrt{2\pi\sigma^2}$.

Hint 1: Separate the integral into a product of two integrands, the first of which (involving $d\theta$) is constant, so is easy.

Hint 2: If $u = e^{-r^2/2\sigma^2}$ then $du/dr = -\frac{1}{\sigma^2}re^{-r^2/2\sigma^2}$, so the second integral is also easy (since $\int u'(r)dr = u(r)$).

3 Reducing the cost of linear regression for large n, small m

The ridge method is a regularized version of least squares with objective function:

$$\min_{\mathbf{w}\in\mathbb{R}^n} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$$

Here λ is a scalar, the input matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$ and the output vector $\mathbf{y} \in \mathbb{R}^m$. The parameter vector $\mathbf{w} \in \mathbb{R}^n$ is obtained by differentiating the cost function, yielding the *normal equations*

$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_n)\mathbf{w} = \mathbf{X}^T\mathbf{y},$$

where $\mathbf{I}_n$ is the $n\times n$ identity matrix. The predictions $\hat{\mathbf{y}} = \hat{\mathbf{y}}(\mathbf{X}_*)$ for new test points $\mathbf{X}_* \in \mathbb{R}^{m^*\times n}$ are obtained by evaluation the hyperplane

$$\hat{\mathbf{y}} = \mathbf{X}_*\mathbf{w} = \mathbf{X}_*(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_n)^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{H}\mathbf{y}.$$

The matrix $\mathbf{H}$ is known as the *hat matrix* because it puts a "hat" on y.

1. Show that the solution can be written as $\mathbf{w} = \mathbf{X}^T\tilde{\mathbf{w}}$, where $\tilde{\mathbf{w}} = \lambda^{-1}(\mathbf{y} - \mathbf{X}\mathbf{w})$.

2. Show that $\tilde{\mathbf{w}}$ can also be written as follows: $\tilde{\mathbf{w}} = (\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_m)^{-1}\mathbf{y}$ and, hence the predictions can be written as follows:

$$\hat{\mathbf{y}} = \mathbf{X}_*\mathbf{w} = \mathbf{X}_*\mathbf{X}^T\tilde{\mathbf{w}} = [\mathbf{X}_*\mathbf{X}^T]([\mathbf{X}\mathbf{X}^T] + \lambda\mathbf{I}_m)^{-1}\mathbf{y}.$$

(This an *awesome trick* because if $m = 20$ patients with $n = 10,000$ gene measurements, the computation of $\tilde{\mathbf{w}}$ only requires inverting the $m\times m$ matrix, while the direct computation of $\mathbf{w}$ would have required the inversion of an $n\times n$ matrix.)

4 Logical Gates Using Perceptrons

Recall that a perceptron with input features $x_1, \dots, x_n$, weights $w_1, \dots, w_n$ and bias w_0 outputs the value:

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

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.

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?

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

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

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

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