Amgad Ahmed

RankNet

Pair-Wise Model

Pair-Wise Algorithms

This type of ranker solves the approximation of the ranking task. It tries to optimize pairwise ordering among pairs. This task is more general then ordering within the list only. We can come up with a situation where such ordering holds U1 < U2, U2 < U3, U3 < U1, and we would like the ranker to be able to learn such ordering. In reality we would not be able to learn the correct ordering entirely and the ranker itself learns an approximation of correct ordering. In general, every algorithm models this approximation differently. The pair-wise algorithm used here is RankNet.

RankNet

In RankNet, the underlying model that we used is a deep neural network. The training is done as follows. The training data is partitioned by query. At a given point during training, RankNet maps an input feature vector x ∈ Rn to a number f(x). For a given query, each pair of documents with different labels is chosen, and each such pair (with feature vectors xi and xj) is presented to the model, which computes the scores si = f(xi) and sj = f(xj). Let Ui > Uj denote the event that Ui should be ranked higher than Uj because Ui has more shares than Uj for this query. The two outputs of the model are mapped to a learned probability that Ui should be ranked higher than Uj via a sigmoid function. The main idea behind using a sigmoid function is that we need an easily differentiable function that models the ordering and can be optimized by its derivative at the same time. A sigmoid function is very well-know and often used in area of neural networks.

$$P_{ij} \equiv P(U_i \triangleright U_j) \equiv \frac{1}{1+e^{-\sigma(s_i - s_j)}}$$

We then apply the cross entropy cost function, which penalizes the deviation of the model output probabilities from the desired probabilities. Cross entropy is used to measure the distance between the two probability vectors. let P¯ i j be the known probability that training document Ui should be ranked higher than training document Uj . Then the cost is:

$$C = -\bar{P}_{ij} \log P_{ij} - (1 - \bar{P}_{ij}) \log(1 - P_{ij})$$

For a given query, let Sij ∈ {0,±1} be defined to be 1 if document i has been labeled to be more relevant than document j, −1 if document i has been labeled to be less relevant than document j, and 0 if they have the same label. This is calculated from the known ordering in the training data.

$$\bar{P}_{ij} = \frac{1}{2}(1 + S_{ij})$$

Given the last two equations, the cost function can be calculated as:

$$C = \frac{1}{2}(1 - S_{ij})\sigma(s_i - s_j) + \log\left(1 + e^{-\sigma(s_i - s_j)}\right)$$

It is easy to see that cost function is larger when Ui > Uj while Uj is actually known to have more shares. And the idea of learning via gradient descent is to update the weights (model parameters) with the gradient of the cost function.

$$\mathrm{w_k} \rightarrow w_k - \eta\left(\frac{\partial C_{ij}}{\partial y_i}\frac{\partial y_i}{\partial w_k} + \frac{\partial C_{ij}}{\partial y_j}\frac{\partial y_j}{\partial w_k}\right)$$

Then a neutral network will be used to optimize the ranking model to minimize the cost function with the back-prop equations. Typically, the accuracy of the trained model will be affected by learning rate, the number of hidden layers and the number of hidden nodes per layer.   While a typical artificial neural network computes this cost by measuring the difference between the network's output values and the respective target values, RankNet computes the cost function by measuring the difference between a pair of network outputs. RankNet attempts to minimise the value of the cost function by adjusting each weight in the network according to the gradient of the cost function. This goal is accomplished through the use of the backpropagation algorithm. The RankNet algorithm receives as input the parameter epochs, which is the number of iterations that are used in the process of providing the network with an input and updating the network's weights, and the parameter hiddenNodes and hidden layers, which corresponds to the number of nodes that are in the network's hidden layers and the number of hidden layers in the network respectively. If there are too few nodes, then we can underfit the data. On the other hand, if there are too many nodes, then we can overfit the data, and the resulting network will not generalise well