1.

You have decided to form a search-engine based start-up. You've collected a dataset of n plain-text search-engine queries $q_1, \ldots, q_n$. For each query q_i, you have also collected the plain-text from the top 1000 web-pages people have subsequently visited, along with the number of total clicks each web-page has received. Thus, for a single plain-text query q, you have

$$\mathcal{D}(q) = (d_1, m_1), \ldots, (d_{1000}, m_{1000}),$$

where d_j is the plain-text for the j'th web-page, and where m_j is a non-negative (integral) click-count for d_j. You may assume that no m_j are repeated for any of the d_j.

For example, for the query q = "restaurants in Harvard Square" you may have collected $\mathcal{D}(q)$ as:

- ("Yelp's list of the best Harvard Square...", 852)
- ("Al's Harvard Square Cafe...", 589)
- etc

Taking the click-count m_j as a proxy for relevance, you are now interested in being able to learn a function that ranks sets of web-pages for which we do *not* have click-counts in terms of their relevance to a given query.

You decide to tackle this problem by learning a function $s(q, d)$ that scores the relevance of a web-page d for a query q. This way, at test time, you can simply score each web-page's relevance independently, and then rank them in terms of these scores.

(a) Using the intuition that we want $s(q, d_j) > s(q, d_k)$ if and only if $m_j > m_k$ — that is, that we want the relevance score of d_j for q to exceed the score of d_k for q if and only if in our training data d_j has more clicks than d_k — write down a loss function $\mathcal{L}$ in terms of the $s(q, d_i)$ that will give high loss if we ever have $s(q, d_j) > s(q, d_k)$ but $m_j \leq m_k$. How many terms are there in $\mathcal{L}$?

$$\mathcal{L} = \sum_{j=1}^{1000} \sum_{k:m_j > m_k} \max\{0, 1 - s(q, d_j) + s(q, d_k)\}$$

A log-loss objective is also fine for this. There are $\binom{1000}{2}$ terms in $\mathcal{L}$.

(b) Write down the derivative of $\mathcal{L}$ wrt a single score $s(q, d_j)$. It may be helpful to use the indicator function $\mathbf{1}(p)$ defined as

$$\mathbf{1}(p) = \begin{cases} 1 & \text{if } p \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial \mathcal{L}}{\partial s(q, d_j)} = - \sum_{k:m_j > m_k} \mathbf{1}(s(q, d_j) < s(q, d_k) + 1) + \sum_{l:m_j < m_l} \mathbf{1}(s(q, d_l) < s(q, d_j) + 1)$$

(c) Using ideas from multi-class classification, write down a loss-function $\mathcal{L}'$ that is similar to $\mathcal{L}$, but which has significantly fewer terms because it looks for only the largest constraint violation.

$$\mathcal{L}' = \sum_{j=1}^{1000} \max_{k:m_j > m_k} \left(\max\{0, 1 - s(q, d_j) + s(q, d_k)\}\right)$$

A log-loss objective could also work, although is not the most natural.

$$\frac{\partial \mathcal{L}'}{\partial s(q, d_j)} = -\mathbf{1}(s(q, d_j) < \max_{k:m_j > m_k} s(q, d_k) + 1) + \sum_{l:j = \arg\max_{k:m_k < m_l} s(q, d_k)} \mathbf{1}(s(q, d_l) < s(q, d_j) + 1))$$

(d) Suppose you realize that instead of ranking all 1000 possible web-pages correctly, it's actually only important to your customers that you rank the 100 most relevant web-pages correctly, since they ignore everything else. Is the loss function you proposed in either of the previous two answers still appropriate? If not, how might you change it?

No, because it cares as much about score-inversions at the end of the ranked-list as at the beginning. To change the objective, it might be reasonable to just forget about $j > 100$ in the outer summation, or to weight the loss of each pair such that pairs containing earlier documents have higher weight than others.

---

2. (This question inspired by "A Dual Embedding Space Model for Document Ranking" (Mitra et al. 2016))

Now that we have some reasonable loss functions for training $s(d, q)$, let's think about how to define it. Let's view each query q as a sequence of one-hot vectors $\boldsymbol{x}_1^0, \ldots, \boldsymbol{x}_{|q|}^0$, where $|q|$ is the length of q. We will view each document d analogously.

We have at our disposal a lookup-table of word-embeddings trained with word2vec. Recall that word2vec uses the following model (for CBOW training):

$$\hat{\boldsymbol{y}} = \text{softmax}((\frac{\sum_i \boldsymbol{x}_i^0 \boldsymbol{W}^0}{d_{\text{win}} - 1})\boldsymbol{W}^1)$$

- $\boldsymbol{x}_i^0 \in \mathbb{R}^{1 \times d_0}$ input words one-hot vectors .
- $\boldsymbol{W}^0 \in \mathbb{R}^{d_0 \times d_{\text{in}}}$; $d_0 = |\mathcal{V}|$, word embeddings
- $\boldsymbol{W}^1 \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$; $d_{out} = |\mathcal{V}|$ output embeddings

(a) Using $\boldsymbol{W}^0$ as your embedding matrix, propose the simplest model for s that you can think of.

$$s(d, q) = \left(\frac{1}{|q|} \sum_{\boldsymbol{x}_i^0 \in q} \boldsymbol{x}_i^0 \boldsymbol{W}^0 \right) \left(\frac{1}{|d|} \sum_{\boldsymbol{x}_i^0 \in d} \boldsymbol{x}_i^0 \boldsymbol{W}^0 \right)^T$$

You can also concatenate the averaged embeddings and put them through an MLP.

(b) After training $s(d, q)$ as defined above, you test it on some very simple one-word queries, and some one-sentence web-pages. After issuing the query "Harvard," you are surprised to find that your scoring function gives the following two sentences roughly the same relevance score

1. "Harvard University is a private Ivy League research university in Cambridge, Massachusetts..." (Wikipedia)
2. "A physics prodigy from Chevy Chase, Dr. Toll studied at Yale and Princeton..." (Washington Post)

What's going on here?

CBOW training forces words that frequently appear in the same contexts (i.e., that are functionally similar) to have similar context-embeddings, so "Harvard" will have a very close embedding to the context-embeddings of other words of the same type, such as "Yale" and "Princeton."

(c) You find that if you use word2vec's "output" embeddings $\boldsymbol{W}^1$ to embed the document's words, but $\boldsymbol{W}^0$ to embed the query's words you no longer have the issue described above. Why not?

Because the "output" word removed from a window will rarely be of the same type as its surrounding words, CBOW training encourages closeness between "input" and "output" embeddings that have the same topic, rather than type. Note that the choice of $\boldsymbol{W}^1$ for the document and $\boldsymbol{W}^0$ for the query was arbitrary; we would expect $\boldsymbol{W}^0$ for the document and $\boldsymbol{W}^1$ for the query to work roughly as well.
