COMP0084 Coursework 1

Anonymous ACL submission

Abstract

None

1 Task 1: Evaluating Retrieval Quality

1.1 Text Processing

We preprocess passages/queries in two files: *validation_data.tsv* and *train_data.tsv* by following steps:

1. remove url.
2. lower characters.
3. remove non alpha characters.
4. tokenization by Python package NLTK.

1.2 BM25 Model

BM25 Model with parameters implemented in the Coursework 1 is used to retrieval the top passages for each query. Here we retrieval top 3, 10 and 100 passages from 1000 passages. The complete result of BM25 model is in *bm25_raw_top1000.tsv*, while we retrieval top 3 passages for each query in *bm25_ordered_top3.tsv*, top 10 passages in *bm25_ordered_top10.tsv* and top 100 passages in *bm25_ordered_top100.tsv*.

1.3 Metrics

1.3.1 Average Precision (AP)

AP is the average precision of relevant passages for a query.

$$AP = \frac{\sum_{k=1}^{n} P(k) \times rel(k)}{N} \quad (1)$$

N: number of relevant passages for the query.
k: rank of the passage.

1.3.2 Mean AP

Mean AP is the average of AP over all queries.

$$mAP = \frac{\sum_{q=1}^{N_q} AP_q}{N_q} \quad (2)$$

q: the q_{th} query

1.3.3 Discounter Cumulative Gain (DCG)

DCG is the total gain accumulated at a particular rank p.

$$DCG_q = \sum_{i=1}^{q} \frac{2^{rel_i} - 1}{\log_2(i+1)} \quad (3)$$

p: particular ranking length
i: the ith passage
rel_i: relevance score

1.3.4 Normalized DCG (NDCG)

Normalizes DCG against the best possible DCG result (the perfect ranking) for a query.

$$NDCG_q = \frac{DCG_q}{optDCG_q} \quad (4)$$

1.3.5 Mean NDCG

Mean NDCG is the average of NDCG over all queries.

$$mNDCG = \frac{\sum_{q=1}^{N_q} NDCG_q}{N_q} \quad (5)$$

N_q: number of queries
The results of the metrics of BM25 model are in Table 1.

Table 1: Metrics of BM25 model

BM25	Cutoff	mAP	mNDCG
Top 3	3	0.1830	0.2007
Top 10	10	0.2250	0.2870
Top 100	100	0.2367	0.3548

2 Task 2: Logistic Regression

2.1 Subsample

The train data set is unbalanced (1% positive, 99% negative). To balance the data set and reduce the training time, we subsample the negative data set

to 20 passages per query and keep all positive passages. For these queries with less than 20 negative passages, we keep all negative passages. The subsampled data set has 95874 passages for 1000 queries.

2.2 Word Embedding

We choose Word2Vec to generate word embedding for each term in the vocabulary. The word embedding is a 100-dimensional vector. We use the pre-trained word embedding and set the window size to 5. The word embedding is trained on train and validation data set separately.

2.3 Logistic Regression

Logistic function:

$$\sigma_{\mathbf{w}}\left(\mathbf{x}_i\right) = \left(1 + e^{-\mathbf{w}^\top \mathbf{x}_i}\right)^{-1} \tag{6}$$

with weight w. The loss function is cross-entropy loss function $\mathcal{J}(\mathbf{w})$:

$$-\frac{1}{n}\sum_{i=1}^{n}\left[y_i \ln\left(\sigma_w\left(x_i\right)\right) + \left(1 - y_i\right)\ln\left(1 - \sigma_w\left(x_i\right)\right)\right] \tag{7}$$

The gradient of the loss function is:

$$\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}_j} = -\frac{1}{n}\sum_{i=1}^{n}\left[x_{i,j}(y_i - \sigma_w\left(x_i\right)\right] \tag{8}$$

We use batch stochastic gradient descent to train the model and stop our training when the validation loss does not decrease for 3 epochs. The word embedding of a query-passage pair given by two vectors with 100 dimensions. We concatenate the two vectors and add a bias term to the model and get a 201-dimensional vector.

We initialize the weight vector with 0, set the batch size to 5000, the tolerance to 1e-8, and the maximum number of epochs to 100. To assess the effect of learning rate on our training, we vary the learning rate from 0.01 to 0.0001 and plot the training loss curve in Fig 1. A larger learning rate results in faster convergence and smaller train loss in the initial epoch. However, all the learning rates converge to the same loss value (0.2) after 100 epochs, except for 0.0001, which still shows no sign of convergence. To confirm convergence, we train the model with 0.0001 for 1000 epochs and present the results in Fig 2. The loss converges to 0.2 within 1000 epochs.

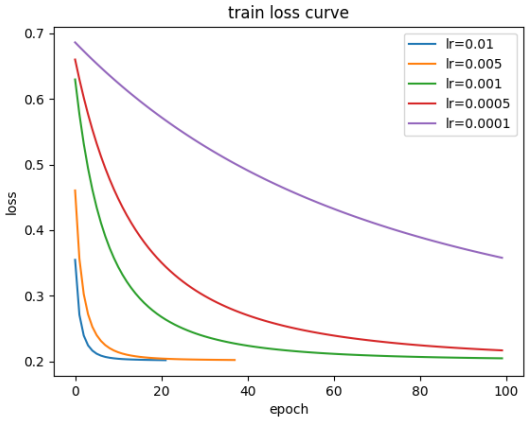

Figure 1: Training loss curve

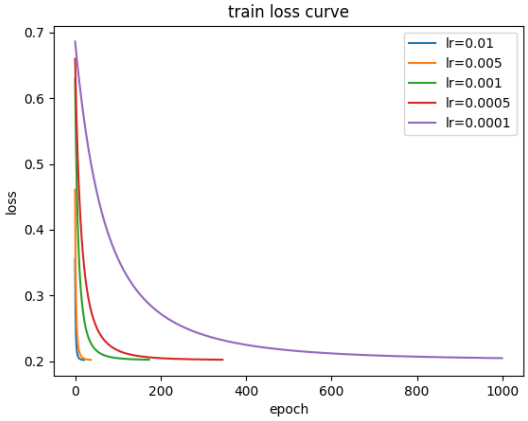

Figure 2: Training loss curve

We decide to use a learning rate of 0.005, batch size of 5000, and epoch number of 100 for the remaining training. Table 2 displays the metrics of the model on the validation set.

We set the cutoff to 100 and apply the model to predict the relevance scores of passages for each query in the file *candidate_passages_top100.tsv*. We save the top 100 highest-scoring passages for each query in a file called *LR.csv*.

Table 2: Metrics of LR model

LR	Cutoff	mAP	mNDCG
Top 100	100	0.0090	0.1254

3 Task 3: LambdaMART Model

3.1 Model

Queries and passages is embeded in the same way as in Task 2. However, we replace the bias term with the relevance of the pair passage-query. The input vector is the same as in Task 2 (embedding vector with 201 dimensions) and the label is the relevance of pair passage-query, 0 or 1. The model is trained using the LambdaMART algorithm using some hyper-parameters.

3.2 Hyper-parameter Tuning

The objective of the model is *rank:ndcg* which use LambdaMART to perform list-wise ranking where Normalized Discounted Cumulative Gain (NDCG) is maximized. We use *gbtree* as the booster. The hyper-parameters are tuned using the validation set. The hyper-parameters are learning rate, number of gradient boosted trees and maximum tree depth for base learners. The metrics mAP and mNDCG of top 100 passages are used to evaluate the model. The grid search method is used to find the best hyper-parameters among the following values: learning rate = 0.0005, 0.001, 0.005, 0.01, 0.05; number of gradient boosted trees = 100, 200, 300; maximum tree depth = 5, 6, 7. The results are shown in Table 3.2. The hyper-parameters resulting in the highest mNDCG are learning rate = 0.005, number of gradient boosted trees = 300 and maximum tree depth = 5. The mAP and mNDCG of top 100 passages are 0.0149 and 0.0388 respectively. According to the metrics on the validation data, LambdaMART model with the hyper-parameters above is not as good as previous models.

No.	eta	depths	n_ests	mAP	mNDCG
0	0.0005	5	100	0.0113	0.0301
1	0.0005	5	200	0.0106	0.0301
2	0.0005	5	300	0.0095	0.0295
3	0.0005	6	100	0.0096	0.0292
4	0.0005	6	200	0.0080	0.0289
5	0.0005	6	300	0.0093	0.0308
6	0.0005	7	100	0.0111	0.0289
7	0.0005	7	200	0.0108	0.0277
8	0.0005	7	300	0.0102	0.0282
9	0.0010	5	100	0.0104	0.0313
10	0.0010	5	200	0.0114	0.0316
11	0.0010	5	300	0.0130	0.0332
12	0.0010	6	100	0.0095	0.0315
13	0.0010	6	200	0.0112	0.0307
14	0.0010	6	300	0.0085	0.0293
15	0.0010	7	100	0.0128	0.0315
16	0.0010	7	200	0.0116	0.0306
17	0.0010	7	300	0.0105	0.0316
18	0.0050	5	100	0.0103	0.0340
19	0.0050	5	200	0.0122	0.0337
20	0.0050	5	300	0.0149	0.0388
21	0.0050	6	100	0.0127	0.0327
22	0.0050	6	200	0.0111	0.0291
23	0.0050	6	300	0.0115	0.0302
24	0.0050	7	100	0.0086	0.0333
25	0.0050	7	200	0.0100	0.0340
26	0.0050	7	300	0.0111	0.0360
27	0.0100	5	100	0.0096	0.0296
28	0.0100	5	200	0.0148	0.0367
29	0.0100	5	300	0.0139	0.0369
30	0.0100	6	100	0.0074	0.0265
31	0.0100	6	200	0.0095	0.0287
32	0.0100	6	300	0.0099	0.0305
33	0.0100	7	100	0.0110	0.0336
34	0.0100	7	200	0.0118	0.0321
35	0.0100	7	300	0.0121	0.0331
36	0.0500	5	100	0.0096	0.0320
37	0.0500	5	200	0.0090	0.0324
38	0.0500	5	300	0.0101	0.0339
39	0.0500	6	100	0.0119	0.0349
40	0.0500	6	200	0.0126	0.0358
41	0.0500	6	300	0.0125	0.0359
42	0.0500	7	100	0.0134	0.0352
43	0.0500	7	200	0.0153	0.0393
44	0.0500	7	300	0.0119	0.0381

4 Task 4: Neural Ranking Model

RNN (Recurrent Neural Network) and LSTM (Long Short-Term Memory) are two types of neural network architectures used for various machine learning tasks including natural language processing.

4.1 RNN

The core idea behind RNN is to process sequential data by maintaining a hidden state that encodes information about previous inputs. The same set of weights is applied to each input in the sequence, allowing the network to handle variable-length input sequences. The memory comes from the hidden state of the network. The hidden state is updated at each time step of the input sequence and retains information about the previous inputs that the network has seen. The current input is combined with the previous hidden state using a set of weights, and the resulting output is used to update the hidden state for the next time step.

One of the drawbacks of the RNN architecture is the vanishing gradient problem, where the gradients that are backpropagated through the network become increasingly small, making it difficult to learn long-term dependencies. Additionally, the RNN architecture can be computationally expensive due to the need to compute the hidden state for each input in the sequence.

4.2 LSTM

LSTM is a variant of RNN architecture designed to address the drawbacks of RNN. It introduces memory cells and gating mechanisms that allow the network to selectively remember or forget previous inputs based on the current input and past context. The memory cells are connected to input and output gates, which control the flow of information into and out of the cells. Additionally, there is a forget gate that can selectively erase information from the memory cells. However, LSTM's complex architecture may limit its use in real-time applications and make it more difficult to interpret and analyze.

Considering the time-series related structure and relationships between query and passage, we choose LSTM as our neural ranking model because its ability to captures long term memory and avoids vanishing gradient.

4.3 Data Process

The terms of a query-passage pair is combined by add two terms together and only the first 100 terms will be used (after tokenization, the average number of terms in a query-passage pair is about 112). We choose model *glove-twitter-100* to represent each pair by a 100-dimension vector. For this pair longer than 100 terms, we only use the first 100 terms. For this pair shorter than 100 terms, we use zero vectors to fill the rest. Hence, for a query-passage pair, we have a 100-dimension vector to represent terms and each element in the vector is a 100-dimension vector to represent a term.