Information Retrieval and Data Mining COMP0084 Project Part 2

20031465
University College London
London, UK

1 EVALUATING RETRIEVAL QUALITY.

I implemented the Average Precision metric and normalized Discounted Cumulative Gain (nDCG). The relevance is binary in the provided datasets so I used a linear gain for DCG.

$$\text{DCG@k} = \sum_{i=1}^{k} \frac{\text{rel}_i}{\log_2(i+1)}$$

$$\text{nDCG@k} = \frac{\text{DCG@k}}{\text{IDCG@k}}$$

The formula for average precision:

$$\text{AP} = \sum_{i=1}^{n} P@i * \mathbb{I}[\text{rel}_i = 1]$$

I implemented the above in the vector form by calculating the rolling sum of the ranking relevancy scores divided by the vector of ranks and summed over the elements of resulting vector. For the BM25 model and Inverted Index I used the code from the previous assignment. The text was pre-processed as before: lowercase, tokenize, remove stopwords, stem. The results on the full validation dataset:

Mean AP	0.0523
Mean nDCG@100	0.1181

For the following parts I used a validation dataset for creating the LR.txt, LM.txt, and NN.txt files.

2 EMBEDDINGS

The embeddings were computed using pre-trained Elmo Small from AllenNLP [7] which returns 256 dimensional embeddings. The main advantage of using Elmo compared to other embedding methods such as word2vec or fastText is that it considers the context of the word. Thus, Elmo should produce better embeddings. Another feature of Elmo is that it is a character-based model. It can handle previously unseen words and misspellings. Since Elmo considers the context, the input text is only tokenized. Operations such as stopwords removal or stemming will worsen the quality of the word embeddings. Technical remarks:

- text is only tokenized into words to capture all semantic information,
- data is sorted based on the text length to produce more uniform batches - speeds up inference,
- the data is passed in batches of 64 due to limited RAM,
- sentence embeddings are produced with mean pooling (acknowledges padding).

The training size is too large to process in a reasonable time. Therefore, for each query, I consider all relevant and 100 randomly chosen non-relevant passages. Then unique queries and passages are passed to Elmo to reduce computational complexity. Finally, the embeddings are concatenated based on the qid and pid resulting in an average of 200 passages for each query.

The validation dataset embeddings are saved in a NumPy .npz compressed file. It is represented as a dictionary with a qid as key and a two dimensional NumPy array of pid, relevancy and concatenated query and passage embeddings as values. That way I can load to RAM only data corresponding to a particular qid. It is convenient for ranking and prevents problems with RAM as the full uncompressed file is over 4GB.

3 LOGISTIC REGRESSION

3.1 Model definition

The logistic regression minimizes cross-entropy loss:

$$-\frac{1}{n}\sum_{i=1}^{n}\big(y_i \log(h_i) + (1-y_i)\log(1-h_i)\big)$$

I also added a small constant inside the logarithms to prevent numerical errors. The weights are initialized with a Uniform(0, 0.1) distribution and bias is set to 0. Continuing, the SGD updates are as follows:

$$weights = weights - lr * \frac{1}{n}\sum_{i=1}^{n}\big((h_i - y_i) * x_i\big)$$

$$bias = bias - lr * \frac{1}{n}\sum_{i=1}^{n}(h_i - y_i)$$

Additionally, I implemented a numerically stable sigmoid function:

$$\text{sigmoid}(x) = \begin{cases} \frac{1}{1+e^{-x}} \text{ , if } x \geq 0 \\ \frac{e^x}{1+e^x} \text{ , otherwise} \end{cases}$$

3.2 Learning Rate Study

The model is trained using mini-batch SGD with batch size 64. Furthermore, the batches are class balanced to avoid biased predictions. Each batch is created by sampling 32 examples with a positive label and 32 examples with a negative label. In Figure 1, each epoch corresponds to 5000 batches. We can see that $lr = 0.1$ is too big. St the beginning, the loss is quickly optimized. Then we see no further improvement in the loss. On the other hand, $lr = 0.001$ is too small, as the loss is slowly decreasing. The $lr = 0.01$ resulted in the best performance. For the final training of the model I used a learning

rate scheduler:

$$\text{lr} = 0.01 * 0.98^{\text{epoch}-1}$$

It allows for fast loss optimization at the beginning of the training and ensures further progress in later stages.

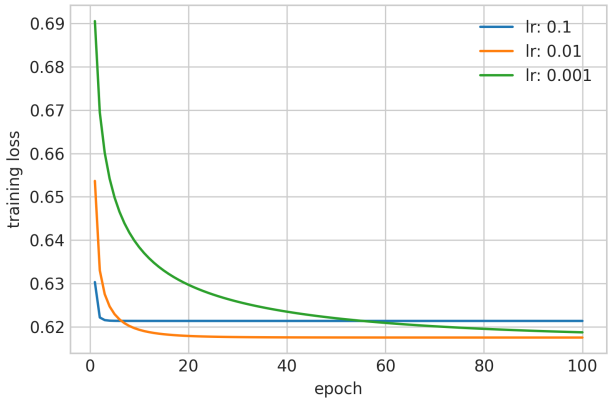

Figure 1: Dependence of learning rate on loss optimization.

3.3 Feature Engineering and Results

The basic set of features includes concatenation of query and passage embeddings. I also considered set of features inspired by InferSent [6] and additionally added a cosine similarity between embeddings:

- concatenation,
- absolute element-wise difference,
- element-wise product,
- cosine similarity.

The training loss is depicted in Figure 2 the comparison of features for Logistic Regression:

The comparison of training losses is depicted in Figure 2 and the performance on the validation dataset is presented below: The

	Mean AP	Mean nDCG@100
Basic Features	0.0215	0.0553
Additional Features	0.0309	0.0821

metrics are calculated on the full validation dataset. We can see that including additional features substantially reduces bias and improves performance of the Logistic Regression model. Logistic Regression assumes that the log-ods are linear in the features. From the empirical evidence, we can reason that the relationship is not linear and handcrafted features are important. We could also consider other features such as:

- BM25 score,
- TF-IDF weights,
- query and passage length.

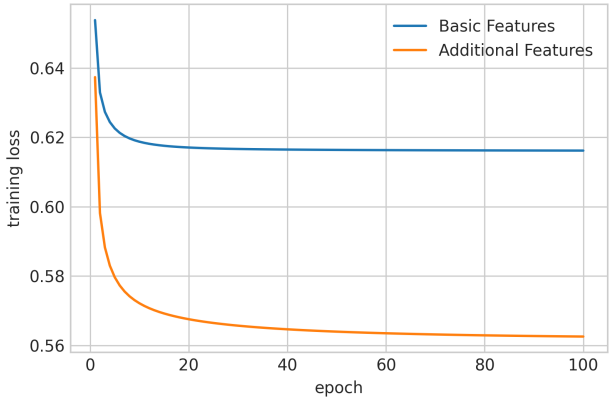

Figure 2: Comparison of training loss.

4 LAMBDAMART

The LambdaMART model also uses additional features as they improved the performance of the previous model.

4.1 Hyperparameter Optimization

The parameter space is 8-dimensional. Therefore grid search approach would be infeasible. Another consideration is to use random grid search as it is known to better explore the parameter space in high dimensions [4]. However, it would still require a substantial number of iterations to reasonably explore the parameter space. Thus, I used a Sequential Model-Based Optimization (SMBO) [3]. It creates a probability model of the objective function to find the most promising hyperparameters. Hence, performs an informed search. The SMBO algorithm outline:

(1) create a surrogate model of the true objective function,
(2) find the best performing hyperparameters on the surrogate,
(3) test the true objective function with the found hyperparameters,
(4) update the surrogate model with new evidence,
(5) repeat steps 2-4 until max iterations.

More precisely, I used a Tree-structured Parzen Estimator (TPE) from the Hyperopt library [5]. To obtain the LambdaMART model using XGBoost library I set the objective to rank:ndcg. Additionally, setting tree_method to gpu_hist allows for GPU compatibility. I modelled the remaining parameters as follows:

- learing_rate: uniform(0.001, 0.1),
- gamma: uniform(1, 3) - minimum loss reduction required to make a split.,
- colsample_bytree: uniform(0.5, 1) - fraction of features for a tree,
- subsample: uniform(0.5, 1) - fraction of observations for a tree,
- reg_alpha: uniform(0, 1) - L1 regularization,
- reg_lambda: uniform(1, 4.5) - L2 regularization,

- min_child_weight: quniform(0, 10, 1) - minimum sum of weights of all observations in a child node; higher values prevent over-fitting,
- max_depth: quniform(0, 20, 1) - max depth of a tree.

The hyperparameter optimization was run for 300 iterations with maximally 1000 boosted rounds. I used a subset of validation dataset by sampling 200 non-relevant passages for each query, keeping all relevant passages. I also used early stopping if the performance of validation loss has not improved for 10 boosted rounds to reduce hyperparameter search time. The found best parameters are listed below:

Parameter	Value
learning_rate	0.0824
cosample_bytree	0.8664
subsample	0.7068
reg_alpha	0.9981
reg_lambda	2.0879
min_child_weight	9
max_depth	12

4.2 Results

I trained the model for 1000 boosted rounds with found parameters on the whole validation dataset. The results are presented in the table below.

Mean AP	0.0365
Mean nDCG@100	0.0901

The LambdaMART model improved upon Logistic Regression.

5 NEURAL NETWORK

5.1 Dense Network

For this part, I will get back to a simple concatenation of the features to see how deep models can learn appropriate representations. For that reason, I build a simple multi-layer perceptron. It has two Linear 512 neuron Layers with ReLU activation followed by Batch Normalization and 20% Dropout. Next, there is a shrinking Layer with ReLU activation that maps from 512 neurons to 128 neurons. Lastly, the output layer maps to a single neuron with Sigmoid activation. The training objective is a Binary Cross-Entropy loss optimized by Adam with lr=0.001. I chose ReLU as the nonlinear activation because it is known to perform well, and it is faster than Tanh or Sigmoid. Continuing, Batch Normalization centres and scales the inputs to the next layer. It makes the network more robust, which can speed up training. Also, Batch Normalization has a slight regularization effect which allows the network to generalize better. Another way to introduce regularization to the network is Dropout, which multiplies the activations by 0 with some probability. It adds noise and hence makes the network more robust. The above architecture was manually tuned. Ideally, we would perform hyperparameter optimization to choose the appropriate number of layers, neurons and optimizer parameters, number of epochs. The model is trained with 20000 balanced batches of 16 samples for 10 epochs

Mean AP	0.0302
Mean nDCG@100	0.0823

as further training does improve performance on the validation dataset. The results on the full validation data are gathered below:

It performed similarly to Logistic Regression on the hand-crafted features, and considerably better than on the concatenated features. So it learned the appropriate representations.

5.2 Pairwise Dense Network

The Pairwise Dense Network has a similar architecture to the Dense Network from the previous subsection. The difference is that it takes two inputs: positive and negative example. Next, it passes each of them through the subnetwork - the same architecture as Dense Network from the previous section without the last Sigmoid activation. Then it subtracts two scalar values and applies Sigmoid activation. The rest of the setting is the same as previously. After training the subnetwork is extracted and used to rank passages from the validation dataset.

Mean AP	0.0333
Mean nDCG@100	0.0855

It improved upon the Dense Network. However, we should perform further variability study to find the confidence of these results. Additionally, we could make the output of the subnetwork multi-dimensional. But then we would have to use the whole network and rank by bubble sort.

5.3 Pairwise Bert Network

Here I fine-tuned msmarco-distilroberta-base-v3 model from Hugging Face repository [11]. It was trained on the MS MARCO ranking dataset [2]. The sentence encoder was trained as a siamese network with cosine similarity, see Figure 3.

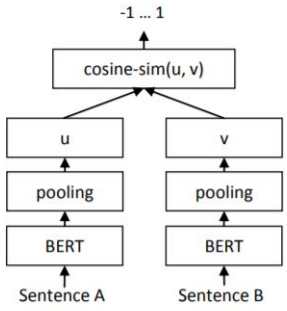

Figure 3: Sentence-Bert architecture [8].

Furthermore, it uses DistillBert, which takes 40% less space and is 60% faster compared to an original Bert model while retaining 97% of its performance [9]. It makes fine-tuning and inference tractable. I created a dataset of balanced bathes with 16 samples per batch. Then I fine-tuned the model for 1 epoch with AdamW optimizer. Next, I added a classifier on top of the embeddings and trained

in a pairwise fashion as previously described in a Pairwise Dense Network section for 8 epochs on 10000 batches of 16 samples.

Mean AP	0.2256
Mean nDCG@100	0.3447

The results on the evaluation dataset are presented in the above table. Bert-based embeddings substantially improved the performance.

5.4 Bert Cross-Encoder

Generally, Cross-Encoders achieve better performance than Bi-Encoders [8]. However, they have fewer applications as they do not output the embeddings. They use as input concatenated sentences such as query and passage separated by a special token. Then the unique embedding for this combination is calculated and passed through a classifier, see Figure 4. This input representation allows finding more sophisticated relations as the multi-head attention is applied jointly across query and passage.

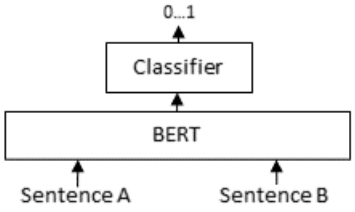

Figure 4: Cross-Encoder architecture [1].

I used a pre-trained cross encoder from Hugging Face repository: cross-encoder/ms-marco-TinyBERT-L-2 [11]. It was also trained on MS MARCO dataset [2]. Additionally, it uses a fast Tiny Bert model [10] with 2 layers, 2 attention heads and 128 dimension size. I further fine-tuned the model for 1 epoch on the training dataset. The final performance of the evaluation dataset is shown in the table below.

Mean AP	0.3683
Mean nDCG@100	0.4915

The cross-encoder model performed the best out of the considered models. Hence, the NN.txt file consists of the passage rankings and scores from Bert Cross-Encoder.

REFERENCES

[1] [n.d.]. SentenceTransformers Documentation. https://www.sbert.net/
[2] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268* (2016).
[3] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, Vol. 24. Neural Information Processing Systems Foundation.
[4] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, 2 (2012).
[5] James Bergstra, Dan Yamins, David D Cox, et al. 2013. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, Vol. 13. Citeseer, 20.
[6] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364* (2017).
[7] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.
[8] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. http://arxiv.org/abs/1908.10084
[9] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).
[10] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962* (2019).
[11] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. HuggingFace's Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).