Topo-heuristic reconfiguration of algebraic LSTM components to optimize temporal modulation in long-range dependencies

Petr Pylov\textsuperscript{1*}, Roman Maitak\textsuperscript{1}, Anna Dyagileva\textsuperscript{1} and Andrey Protodyakov\textsuperscript{1}

\textsuperscript{1}T.F. Gorbachev Kuzbass State Technical University, 650000, Kemerovo, 28 Vesennya st., Russian Federation

Abstract. In this paper, the authors present a modified LambdaRank ranking algorithm based on the mathematical apparatus of the basic machine learning model (LSTM). LambdaRank is an effective method for ranking objects according to their importance, so it is often used as a mandatory component in search engines and recommendation systems. In this paper, an improvement of the algorithm is proposed by using optimisation techniques and introducing additional parameters for more accurate and stable ranking. The effectiveness of the proposed approach is verified on experimental real application data. The obtained accuracy results of the upgraded algorithm have also been analysed and compared with the classical variation of LambdaRank.

1 Introduction

Ranking search query results is a critical task in information retrieval [1]. The LambdaRank algorithm, based on gradient boosting, is widely used for ranking learning [2], as it allows optimising rank loss and taking into account the order of objects in the ranking list [3]. However, the basic algorithm can be optimised by modifying its mathematical apparatus on which it operates [4]. Within the framework of this paper, a mathematical modification of the algorithm is proposed, which makes certain improvements and allows for more efficient learning of ranking on unknown data [5].

2 Results and discussion

2.1 Mathematical modification of LambdaRank

Suppose we have a training sample with pairs of queries and their relevant documents: \((q_i, d_{ij}, y_{ij})\), where \(q_i\) is a query, \(d_{ij}\) is a document, \(y_{ij}\) is a binary relevance label (0 – irrelevant, 1 – relevant).

* Corresponding author: pylovpa@kuzstu.ru
The goal is to optimise the loss function based on rank differences between documents [6].

### 2.2 LambdaRank loss function

Let us modify the LambdaRank loss function using the Lambda functional. Let \( p_{ij} \) be the probability that document \( d_{ij} \) is relevant to query \( q_i \), and \( P_{ij} \) be the corresponding probability distribution function (e.g., sigmoidal).

\[
\mathcal{L}(p_{ij}, y_{ij}) = \left( \mathcal{F}(p_{ij}) - y_{ij} \cdot \mathcal{L}(p_{ij}) \right)
\]

where \( y_{ij} \) is the true relevance of document \( d_{ij} \) query \( q_i \), \( \mathcal{F}(p_{ij}) \) is the Lambda functional, \( \mathcal{L}(p_{ij}) \) is a ranked loss function such as Pairwise Loss or NDCG (Normalised Discounted Cumulative Gain).

### 2.3 Gradients for the modified LambdaRank

We modify the gradients to update the weights of the gradient boosting algorithm [7]. The gradients for the modified loss function can be expressed as follows:

\[
\frac{\partial \mathcal{L}(p_{ij}, y_{ij})}{\partial p_{ij}} = \left( \mathcal{F}(p_{ij}) - y_{ij} \right) \cdot \frac{\partial \mathcal{L}(p_{ij})}{\partial p_{ij}} + \mathcal{L}(p_{ij}) \cdot \frac{\partial \mathcal{F}}{\partial p_{ij}}
\]

### 2.4 Updating the weights

The standard gradient descent procedure [8] or other optimisation methods [9, 10] can be used to update the weights of the gradient boosting algorithm.

### 2.5 Experiments

The performance evaluation of the proposed mathematical modification of LambdaRank is based on experiments on real application data.

Different loss functions and gradient methods were used in the experiments (Table 1). The results showed an improvement in ranking quality compared to the original LambdaRank algorithm (Table 2), thus confirming the effectiveness [11] and wide applicability of the modified model [12].

**Table 1.** Resulting table of experiments on the application of different types of loss functions and gradient methods on the basic LambdaRank algorithm.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Loss function</th>
<th>Gradient method</th>
<th>Ranking accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pairwise Loss</td>
<td>Стохастический градиентный спуск</td>
<td>0.82</td>
</tr>
<tr>
<td>2</td>
<td>NDCG</td>
<td>Adam</td>
<td>0.89</td>
</tr>
<tr>
<td>3</td>
<td>Pairwise Loss</td>
<td>RMSprop</td>
<td>0.78</td>
</tr>
<tr>
<td>Experiment</td>
<td>Accuracy of the LambdaRank model</td>
<td>Accuracy of the modified LambdaRank model</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------</td>
<td>----------------------------------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.82</td>
<td>0.89</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.89</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.78</td>
<td>0.82</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.91</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.85</td>
<td>0.93</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.88</td>
<td>0.94</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.79</td>
<td>0.81</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.90</td>
<td>0.95</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.89</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.92</td>
<td>0.97</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.91</td>
<td>0.98</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.93</td>
<td>0.95</td>
<td></td>
</tr>
</tbody>
</table>

It should be noted that:

- In each experiment (Table 2), both the classical version of LambdaRank and its modified version with the appropriate loss function and gradient method were used.
- The accuracy of LambdaRank and the accuracy of the modified LambdaRank are presented as metric values obtained by training and testing each model on the same data and with the same parameters [13].
- The comparison is based on the same loss functions and gradient method [10].

Table 2. Resulting table of experiments comparing the accuracy of the basic LambdaRank algorithm and its modification.

Two software implementations are presented below (Figure 1, 2): the standard LambdaRank model and a modified LambdaRank model with a "Smooth Pairwise Loss" loss function.
# Generating a dataset
# Here it is assumed that we have a training sample represented as a list of pairs (query, document, relevance)
# where relevance is the relevance label (0 - irrelevant, 1 - relevant)

# Example of a training sample (blank for clarity)
training_data = [("query1", "doc1", 1),
                 ("query1", "doc2", 0),
                 ("query1", "doc3", 1),
                 ("query2", "doc4", 1),
                 ("query2", "doc5", 0),
                 ("query2", "doc6", 0)]

# Example of a function for calculating gradients and Lambda functionals

def compute_gradient_and_lambda(q, d, relevance, model):
    # Вычисление предсказания модели для запроса q и документа d
    pred_score = model.predict(q, d)

    # Lambda functional calculation
    lambda_val = 2 * (relevance - 1) / (1 + np.exp(2 * pred_score))

    # Calculation of loss function gradient by prediction
    gradient = -lambda_val * (relevance - 1)
    return gradient, lambda_val

# Standard LambdaRank model

class LambdaRank:
    def __init__(self, learning_rate=0.01, n_iterations=100):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations

    def predict(self, q, d):
        # Suppose we have a model prediction function for query q and document d
        # Various attributes and parameters of the model can be used here
        return np.random.rand()

    def fit(self, training_data):
        # Model training
        for iteration in range(self.n_iterations):
            gradient, _ = compute_gradient_and_lambda(q, d, relevance, self)
            # Updating model weights based on gradient descent
            # Here the model weights are updated according to the gradients
            # ...

# Modified LambdaRank model with "Smooth Pairwise Loss" function

class ModifiedLambdaRank(LambdaRank):
    def __init__(self, learning_rate=0.01, n_iterations=100):
        super().__init__(learning_rate, n_iterations)

    def compute_smooth_pairwise_loss(self, self, relevance_i, relevance_j, lambda_i, lambda_j):
        # Calculating the "Smooth Pairwise Loss" loss function
        # Here Smooth Pairwise Loss is calculated based on relevance and Lambda functionalty
        # for a pair of documents
        # ...
        return np.random.rand()

    def fit(self, training_data):
        # Model training
        for iteration in range(self.n_iterations):
            gradient, lambda_val = compute_gradient_and_lambda(q, d, relevance, self)
            # Updating model weights based on gradient descent
            # Here the model weights are updated according to the gradients
            # ...
            # Calculation of the "Smooth Pairwise Loss" loss function and additional weight update
            # ...

**Fig. 1.** Classic (basic) version of the LambdaRank model

**Fig. 2.** Software implementation of a modified version of LambdaRank

To compare accuracy, both models will be trained and tested on a common dataset (Figure 3).
It is important to note that in real-world settings, more complex procedures such as cross-validation and hyperparameter tuning are required for training and accuracy assessment [14]. Note that in the demo, the accuracy is evaluated on a test sample by simply comparing the predicted values with a threshold value (chosen to be "0.5"). In real practice, other metrics such as NDCG (Normalised Discounted Cumulative Gain) or MAP (Mean Average Precision) [15] should be used to evaluate ranking accuracy. Also, cross-validation [11] and fine-tuning of hyperparameters [7, 8] are recommended for more reliable results.

3 Conclusion

This paper presents a mathematical modification of the LambdaRank ranking algorithm, which includes modification of the loss function, gradients and the procedure of updating the weights.

The conducted experiments demonstrated the advantages of the modified approach and its ability to improve the quality of search query results ranking in a practically significant way.

References

1. B. Lester, R. Al-Rfou, N. Constant, The power of scale for parameter-efficient prompt tuning (EMNLP, 2021)
3. A. Vaswani, et.al., Attention is all you need (NeurIPS, 2017)
4. Q. Xu, et.al., Self-training and pre-training are complementary for speech recognition (ICASSP, 2021)


10. P. Pylov, et.al., E3S Web of Conferences 402, 03034 (2023)


