Academic Year 2023/2024

Biomedical Informatics

POLITÉCNICA

Machine Learning Ranking List-wise Approach: AdaRank

Training Set Creation & Implementation

Group 11

Andreas Brummer

Lorenzo Leuzzi

Michele Simeone

Project Repository: github.com/BioInformatics-AdaRank

1 Introduction

The order of search results has gained significant importance due to the continuous growth of measurements and information, leading to an increase in query responses. The accuracy and the speed of the sorting is especially crucial in the healthcare sector, where providing patients with prompt and accurate care is of utmost significance.

To achieve this objective, machine learning ranking comes to our aid. There are three primary options: pointwise, pairwise, and listwise. Nowadays, listwise approach was predominantly employed, so we choiced it for our assignment, based on the research paper titled 'AdaRank: A Boosting Algorithm for Information Retrieval'.

Queries such as 'glucose in blood,' 'bilirubin in plasma,' and 'white blood cell count' were optimized using AdaRank and in addition, to complete this brief overview, we employed data augmentation techniques to obtain an expanded version of our dataset.

2 AdaRank: Key Idea

Ideally, a learning algorithm should train a ranking model capable of optimizing performance measures directly in relation to the training data. However, existing methods are limited in that they can only train ranking models by minimizing loss functions that are loosely associated with performance measures. For example, Ranking SVM and RankBoost train ranking models by minimizing classification errors on instance pairs.

To address this limitation, a learning algorithm is proposed within the framework of boosting. This algorithm, referred to as AdaRank, repeatedly constructs 'weak rankers' by adjusting the weights of the training data and ultimately linearly combines these weak rankers to make ranking predictions.

The primary objective of learning is to create a ranking function, denoted as $f : X \to R$. This function allows for the assignment of relevance scores to elements in the corresponding document list for each query. Subsequently, these elements can be ranked based on their scores.

Throughout the learning process, AdaRank maintains a distribution of weights over the queries in the training data. Initially, equal weights are assigned to all queries. In each round, AdaRank increases the weights of queries that are not ranked well by the model created thus far. Consequently, the focus of the next round's learning is on the development of a weak ranker capable of handling the ranking of 'hard' queries.

The advantages offered by AdaRank include ease of implementation, theoretical soundness, efficiency in training, and high accuracy in ranking. Experimental results on four benchmark datasets, indicate that AdaRank can outperform baseline methods such as BM25, Ranking SVM, and RankBoost.

Regarding time complexity, AdaRank operates with a time complexity of $O((k + T)m \cdot n \log n)$, where k represents the number of features, T is the number of rounds, m denotes the number of queries in the training data, and n is the maximum number of documents for queries in the training data.

3 Preparing Dataset

Every dataset used for a listwise ranking approach requires labels that indicate the relevance of each sample to the query under consideration. However, manually labeling even a small dataset with a complete ranking becomes an unfeasible task, as it is incredibly time-consuming and demands extensive knowledge of the subject matter to which the data pertains. Consequently, most datasets categorize the samples into three groups: 'Relevant,' 'Partially Relevant,' and 'Not Relevant.' This approach was also adopted in our experiment, with the three categories encoded as labels 2, 1, and 0.

To label the data, given our limited foundational knowledge of the field of the dataset, we utilized the *LOINC search* accessible via the LOINC website[1]. Specifically, we mapped all the samples from the original dataset that had a LOINC code present in the results of a complete query search to Tier 2. We then repeated

[1] https://loinc.org/search/

the search with a partial query (containing only the terms 'Glucose,' 'Bilirubin,' and 'White Blood') and mapped all the samples found in Tier 1. Finally, all the remaining samples were assigned to Tier 0.

To expedite this task, we developed a Python notebook that, using the Pandas library, automatically inspects and executes the mapping process as described above.

4 Extending LOINC dataset

In the context of biomedical informatics, **data augmentation** is a crucial strategy, particularly when working with datasets of limited size. The expansion of datasets through the addition of extra rows or records is a fundamental step to be able to better represent the desired data distribution through a sample of it.

In our study, we used the *LOINC search*, to effectively augment our dataset. Our ranking system, as discussed in Section 3, is structured as follows: 0, 1, and 2, with 2 indicating the highest degree of relevance. For each given query to populate their respective classes, we conducted comprehensive searches using the results produced from a specific inquiry within the LOINC database as specified before. For example, for class 0, we deliberately selected queries that bore no relevance, serving as representations of entirely disparate medical conditions. For each class we selected 50 of the found items.

Following the retrieval of search results from the LOINC website in the *.csv* format, we used a Python notebook, in particular the *pandas* library to merge the results with the initial *excel* file. Through this integration we aimed to substantially enrich our dataset, thereby enhancing the comprehensiveness of our biomedical informatics analysis.

The aforementioned extension increases our dataset to contain between 215 and 220 samples for each query. However, since the procedure is automated and scalable, it is possible to increase the number of samples at any time.

5 Model Implementation

5.1 AdaRank code

Having acquired a comprehensive understanding of the AdaRank algorithm, the next phase involved its practical implementation through *Python* programming. The high-level algorithm logic can be summarized as follows:

Algorithm 1 AdaRank Pseudocode

$weights \leftarrow 1/m$
for $t = 1, ..., T$ **do**
    create a weak ranker h_t with weighted distribution
    choose α_t
    create the scoring function f_t
    update the weights
end for

The implementation of this algorithm is further detailed in Figure 5.1, where the code and its application can be examined in a more granular manner.

```python
n_queries = np.unique(qid).shape[0]
weights = np.ones(n_queries, dtype=np.float64) / n_queries
```

(a)

```python
weak_ranker_score = []
for j in range(X.shape[1]):
    pred = X[:, j].ravel()
    weak_ranker_score.append(self.scorer(y, pred, qid))
```

(b)

```python
best_weighted_average = -np.inf
best_weak_ranker = None
for fid, score in enumerate(weak_ranker_score):
    if fid in used_fids:
        continue
    weighted_average = np.dot(weights, score)
    if weighted_average > best_weighted_average:
        best_weak_ranker = {'fid': fid, 'score': score}
        best_weighted_average = weighted_average
```

(c)

```python
h = best_weak_ranker
h['alpha'] = 0.5 * (math.log(np.dot(weights, 1 + h['score']) /
                             np.dot(weights, 1 - h['score'])))
weak_rankers.append(h)
```

(d)

```python
score_train = self.scorer(y, np.dot(X, coef), qid)
```

(e)

```python
# update weights
new_weights = np.exp(-score_train)
weights = new_weights / new_weights.sum()
```

(f)

Figure 5.1: Code implementation for a) initialization; b) precomputing features scores for the weak ranker; c) selecting the feature with best weighted score; d) calculating alpha; e) creating the scoring function f_t; f) updating weights

5.2 LETOR benchmark

This version of AdaRank was then tested using the *LETOR: Learning to Rank for Information Retrieval* dataset Qin2010LETOR. This dataset encompasses a total of 46 features employed in the ranking of documents. As illustrated in Figure 5.2, the obtained results, although not exceptionally impressive, provided a tangible indication that the algorithm was functional and capable of delivering ranking outcomes.

Indeed, upon a careful comparison with existing state-of-the-art results, which notably stand at 0.4915, our achieved result of 0.4077 signifies a performance that, while marginally less favorable. It is essential to note that our evaluation employed a slightly smaller test set in comparison to the dataset paper, rendering our results, if anything, somewhat optimistic. This underscores the potential for further refinements and optimizations in the pursuit of even more competitive outcomes.

```python
model = AdaRank(max_iter=max_iter,
                estop=patience,
                verbose=False,
                scorer=NDCGScorer(k=k))

model.fit(X, y, qid)

predictions = model.predict(X_test, qid_test)
for k in (1, 2, 3, 4, 5, 10, 20):
        score = NDCGScorer(k=k)(y_test, predictions, qid_test).mean()
        print('nDCG@{}\t{}'.format(k, score))
```

```
nDCG@1  0.40773809523809523
```

Figure 5.2: Testing

5.3 Extracting Features

The AdaRank algorithm's underlying assumption necessitates that the dataset already encompasses the relevant features for the intended analysis. In our specific case, this assumption did not align with the nature of our LOINC dataset. To facilitate the execution of AdaRank or any other subsequent algorithm, a crucial preliminary step was undertaken: the extraction of the requisite features from the collection of documents, in our case the long LOINC names, concerning their relevance to the query. We diligently crafted code tailored to this purpose.
The considered metrics are the following:

- **IDF** (Inverse Document Frequency):

$$\text{IDF}(t) = \log\left(\frac{N}{\text{df}(t)}\right)$$

where N is the total number of documents and $\text{df}(t)$ is the number of documents containing term t.

- **Cosine Similarity**:

$$\text{Cosine Similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

where $\mathbf{A}$ and $\mathbf{B}$ are vectors representing the term frequencies or TF-IDF values of two documents.

- **Jaccard Score**:

$$\text{Jaccard Score}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

where A and B are sets representing the terms in two documents.

- **BM25** (Best Matching 25):

$$\text{BM25 Score}(D, Q) = \sum_{i=1}^{n} \left(\frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avg\_doc\_length}}\right)} \right) \cdot \text{IDF}(q_i)$$

where D is the document, Q is the query, n is the number of query terms, $f(q_i, D)$ is the frequency of term q_i in the document D, k_1 and b are tuning parameters, and $\text{IDF}(q_i)$ is the Inverse Document Frequency of term q_i.

5.4 LOINC Experiments and Results

In this study, we conducted a comprehensive assessment of the AdaRank algorithm implementation, utilizing the LOINC dataset, subsequent to the extraction of pertinent features. The dataset was procured from an Excel file generated for this purpose, and a deliberate partitioning was performed, allocating 85% of the data for training and reserving 15% for testing. During this process, we diligently employed both shuffling and stratification techniques to ensure equitable representation of classes. The primary objective of this endeavor was to underscore the inherent limitations of the original dataset, notably its size, and to investigate the potential benefits of an extension process in enhancing the algorithm's capacity to discern the $(document, query) \rightarrow rank$ mapping effectively.
Our empirical findings, as illustrated in the comparative results Table 5.1, reveal a notable and substantial enhancement in test performance when employing the extended dataset for both training and testing. This noteworthy improvement underscores the pivotal role of dataset expansion in facilitating the algorithm's learning process. Furthermore, it is important to underline that these results, while promising, should be interpreted with caution. The observed improvements in performance, as indicated earlier, may be subject to limitations stemming from the relatively diminutive size of the test set.

	Original	**Extended**
nDCG@1	0	0.6667
nDCG@2	0	0.6667
nDCG@3	0	0.6145
nDCG@4	0.1186	0.5672
nDCG@5	0.1541	0.5749
nDCG@10	0.295	0.7117
nDCG@20	0.295	0.8048

Table 5.1: Comparison between the results obtained while training and testing using the Original LOINC dataset versus the Extended one