Indian Institute of Technology Delhi

Department of Computer Science and Engineering

COL 764: Assignment 3

Learning to Rank

Submitted by

Student Name: Rohan Chaturvedi
Registration Number: 2022MT11262

Course Instructor

Srikanta Bedathur

Submission Date: 25/10/2024

Contents

1 Introduction

1.1 Problem Overview

In this assignment, we explore the application of machine learning models for Learning to Rank (LTR) using query-document feature datasets TD2003 and TD2004. The implementation focuses on three regression models and their performance evaluation using standardized metrics.

1.2 Objectives

- Implementation of three distinct regression models for ranking
- Evaluation using nDCG metrics
- Performance optimization and hyperparameter tuning
- Comparative analysis of model behaviors

2 Methodology

The Learning to Rank (LTR) task is formulated as a pointwise ranking problem, where the goal is to learn a function that maps query-document feature vectors to relevance scores. Given a set of queries Q and documents D, each query-document pair (q, d) is represented by a feature vector $x_{q,d} \in \mathbb{R}^n$, and the objective is to learn a scoring function $f : \mathbb{R}^n \to \mathbb{R}$ that assigns relevance scores matching the ground truth labels.

2.1 Support Vector Regressor (SVR)

Support Vector Regression aims to find a function $f(x)$ that has at most ϵ deviation from the actual targets y for all the training data, while remaining as flat as possible. The SVR model utilizes the ϵ-insensitive loss function:

$$L_\epsilon(y, f(x)) = \max(0, |y - f(x)| - \epsilon)$$

The optimization problem is formulated as:

$$\min_{w,b,\xi,\xi^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

subject to:

$$\begin{aligned} y_i - (w^T\phi(x_i) + b) &\leq \epsilon + \xi_i \\ (w^T\phi(x_i) + b) - y_i &\leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* &\geq 0 \end{aligned}$$

The use of the RBF kernel allows the model to handle non-linear relationships by implicitly mapping the input features to a higher-dimensional space:

$$K(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$$

2.2 Gradient Boosting Regressor (GBR)

Gradient Boosting constructs an additive model in a forward stage-wise fashion. At each stage, the model fits a base learner (decision tree) to the negative gradient of the loss function. The model can be expressed as:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

where:

- $F_m(x)$ is the model at iteration m
- $h_m(x)$ is the base learner (regression tree)
- γ_m is the step length

The loss function being minimized is the Mean Squared Error:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - F(x_i))^2$$

At each iteration, the algorithm:

1. Computes the negative gradient of the loss function
2. Fits a regression tree to the negative gradient
3. Performs line search to determine the optimal step size
4. Updates the model

2.3 Multi-Layer Perceptron (MLP)

The Multi-Layer Perceptron is implemented as a deep neural network that processes input through multiple non-linear transformations. The network consists of an input layer, multiple hidden layers, and an output layer, with the following forward propagation equation:

$$h^{(l)} = \text{ReLU}(W^{(l)}h^{(l-1)} + b^{(l)})$$

where:

- $h^{(l)}$ is the output of layer l
- $W^{(l)}$ is the weight matrix for layer l
- $b^{(l)}$ is the bias vector for layer l
- ReLU is the activation function: $f(x) = \max(0, x)$

The network is trained to minimize the mean squared error loss with L2 regularization:

$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha\sum_{l=1}^{L}\|W^{(l)}\|_F^2$$

where:

- First term represents the prediction error
- Second term is the L2 regularization to prevent overfitting

- $\|\cdot\|_F$ denotes the Frobenius norm

The architecture processes the features through multiple hidden layers with ReLU activation functions, culminating in a linear output layer for regression. The use of ReLU activation introduces non-linearity while avoiding the vanishing gradient problem common in deeper networks. The Adam optimizer is employed for training, which adapts the learning rate for each parameter, providing efficient convergence.

3 Hyperparameters

3.1 Approach

To identify the optimal hyperparameters for the machine learning models used in this ranking task, a systematic search approach was employed. The method aimed to explore a range of values for each hyperparameter and evaluate the model performance for each configuration. The search was guided by an iterative process, where each trial selected hyperparameters, the model was trained and tested, and performance metrics (like nDCG) were evaluated. The hyperparameters for each model were chosen from predefined ranges, and the values that maximized the performance metrics were selected.

3.2 Hyperparameter Configuration

The hyperparameters were tuned separately for each dataset (TD2003 and TD2004) across the models. Below are the final chosen values along with the ranges they were selected from.

3.2.1 SVR Parameters

The SVR model was optimized over the following hyperparameters:

- Kernel: `['rbf', 'linear', 'poly']`
- C: `[0.01, 0.1] (logarithmic scale)`
- ϵ: `[5e-3, 1] (logarithmic scale)`
- γ: `[0.01, 0.1] (logarithmic scale)`

For TD2003:

- Kernel: RBF
- C: 0.0602
- ϵ: 0.0114
- γ: 0.0107

For TD2004: The SVR model followed a similar procedure, and the final parameters selected for TD2004 are documented in Table **??**.

3.2.2 GBR Parameters

For the Gradient Boosting Regressor (GBR), the following hyperparameters were explored:

- Estimators: `[50, 300]`
- Learning Rate: `[0.01, 1] (logarithmic scale)`

- Max Depth: `[3, 10]`
- Min Samples Split: `[2, 20]`
- Min Samples Leaf: `[1, 15]`
- Loss: `['squared_error', 'ls', 'lad', 'huber', 'quantile']`

For TD2003:

- Estimators: 154
- Learning Rate: 0.0792
- Max Depth: 6
- Min Samples Split: 4
- Min Samples Leaf: 6

For TD2004:

- Estimators: 154
- Learning Rate: 0.0792
- Max Depth: 6
- Min Samples Split: 4
- Min Samples Leaf: 6

3.2.3 MLP Parameters

The Multi-Layer Perceptron (MLP) model was optimized with the following hyperparameters:

- Hidden Layer Sizes: `['(100, 100)', '(100)', '(50)', '(100,50)', '(50,100)', '(150)']`
- Activation: `['relu', 'tanh']`
- Alpha: `[1e-5, 5e-2] (logarithmic scale)`
- Learning Rate Init: `[1e-4, 1e-2] (logarithmic scale)`

For TD2003:

- Hidden Layer Sizes: (100, 100)
- Activation: relu
- Alpha: 0.040
- Learning Rate Init: 0.00015

For TD2004:

- Hidden Layer Sizes: (100, 50)
- Activation: tanh
- Alpha: 0.045
- Learning Rate Init: 0.0002

Each model's hyperparameters were selected through a process that balanced generalization ability with model complexity, ensuring robust performance across different ranking datasets.

4 Results and Analysis

4.1 Performance Metrics

Table 1: Comprehensive Model Performance Across Datasets and Folds

Dataset	Fold	Model	Train nDCG			Eval nDCG			Test nDCG			Train	Test
			@5	@10	@100	@5	@10	@100	@5	@10	@100	Time(s)	Time(s)
TD2003	1	SVR	0.82	0.77	0.78	0.39	0.33	0.30	0.24	0.20	0.26	2.25	0.813
		GBR	0.89	0.85	0.88	0.50	0.45	0.42	0.28	0.28	0.34	43.57	0.03
		MLP	0.64	0.61	0.71	0.47	0.39	0.41	0.23	0.22	0.35	30.70	0.04
	2	SVR	0.84	0.76	0.75	0.25	0.20	0.29	0.24	0.22	0.28	2.7	0.74
		GBR	0.91	0.89	0.88	0.30	0.28	0.35	0.23	0.24	0.30	49.09	0.04
		MLP	0.73	0.70	0.76	0.36	0.33	0.44	0.39	0.38	0.48	35.76	0.06
	3	SVR	0.82	0.73	0.72	0.33	0.28	0.35	0.13	0.19	0.28	3.61	0.84
		GBR	0.86	0.83	0.84	0.16	0.17	0.27	0.14	0.14	0.24	51.83	0.03
		MLP	0.71	0.64	0.71	0.46	0.39	0.49	0.43	0.46	0.52	42.33	0.05
	4	SVR	0.85	0.75	0.75	0.13	0.18	0.28	0.29	0.30	0.37	3.53	0.93
		GBR	0.93	0.87	0.88	0.25	0.28	0.38	0.41	0.43	0.48	46.92	0.03
		MLP	0.74	0.66	0.76	0.41	0.42	0.49	0.34	0.33	0.39	41.92	0.05
TD2004	1	SVR	0.59	0.60	0.65	0.33	0.31	0.35	0.40	0.36	0.42	2.54	1.06
		GBR	0.89	0.89	0.91	0.37	0.39	0.52	0.35	0.35	0.46	106.97	0.06
		MLP	0.46	0.48	0.59	0.47	0.49	0.58	0.47	0.44	0.53	16.03	0.07
	2	SVR	0.63	0.64	0.69	0.40	0.36	0.40	0.21	0.23	0.31	3.67	1.22
		GBR	0.93	0.93	0.95	0.35	0.39	0.47	0.26	0.30	0.42	108.03	0.05
		MLP	0.49	0.50	0.64	0.48	0.47	0.54	0.38	0.42	0.54	9.10	0.04
	3	SVR	0.64	0.63	0.66	0.22	0.22	0.30	0.31	0.32	0.47	3.40	1.36
		GBR	0.91	0.90	0.91	0.24	0.26	0.40	0.39	0.43	0.52	107.34	0.06
		MLP	0.44	0.46	0.57	0.36	0.40	0.52	0.45	0.49	0.64	10.83	0.07
	4	SVR	0.64	0.64	0.68	0.32	0.34	0.43	0.17	0.22	0.30	3.15	1.48
		GBR	0.85	0.85	0.86	0.33	0.37	0.53	0.36	0.38	0.52	113.01	0.04
		MLP	0.46	0.47	0.57	0.45	0.49	0.61	0.33	0.38	0.49	8.37	0.04

4.2 Analysis

- MLP performed much better than the other two, with GBR being better than SVR
- while SVR was fastest to compute, MLP and GBR had higher computation time

5 Time Optimisation

In this section, we discuss the strategies employed to improve the runtime efficiency of the models, focusing on hyperparameter tuning, subsampling techniques, and model simplification.

5.1 Hyperparameter Tuning for Efficiency

To balance model performance with computational efficiency, certain choices were made regarding hyperparameters:

- **SVR Parameters**: The RBF kernel was chosen after testing alternatives like linear and polynomial kernels, as it provided a good balance between performance and computation time. Additionally, the regularization parameter C was kept relatively low (0.0602) to prevent overfitting, which helped in reducing training time. A small value of ϵ (0.0114) was also selected to control the margin for error, limiting unnecessary iterations during training.
- **GBR Parameters**: In Gradient Boosting, the number of estimators was restricted to 154 to control training time, as higher values significantly increase runtime. A moderate learning rate of 0.0792 was chosen to ensure convergence without excessive iterations. Additionally, controlling the maximum depth to 6 helped limit the complexity of individual trees, thus speeding up training. Parameters like `min_samples_split` and `min_samples_leaf` were tuned to avoid creating overly complex trees that would slow down training.
- **MLP Parameters**: For the Multi-Layer Perceptron, the number of neurons in hidden layers was minimized to the configurations `(100,100)` and `(50,100)` to reduce the number of computations in each forward and backward pass. The activation function `tanh` was selected for its efficiency during training in TREC 2004 dataset, as it involves fewer complex computations compared to alternatives like `relu`. A lower learning rate initialization and regularization via `alpha` helped maintain faster convergence while avoiding excessive computations during each iteration.

5.2 Subsampling for Speed

To further reduce computation time, subsampling techniques were used during the hyperparameter search and model evaluation process:

- **Dataset Subsampling**: Instead of using the entire dataset for each trial in hyperparameter tuning, a subsample of the data was used. This reduced the time required for each iteration while still providing a representative sample for tuning. By carefully selecting smaller training subsets, it was possible to maintain the quality of the hyperparameter selection while significantly reducing runtime.

By leveraging these time optimization strategies, the training process was significantly accelerated, making it feasible to explore multiple models and hyperparameter configurations within a limited time.

6 BONUS TASK

In this section, we describe the steps taken to address the bonus task by incorporating dimensionality reduction and data preprocessing techniques, aimed at improving both speed and performance.

6.1 Principal Component Analysis (PCA)

To manage the high-dimensional feature space and reduce computational costs, we applied **Principal Component Analysis (PCA)** as a dimensionality reduction technique. By using

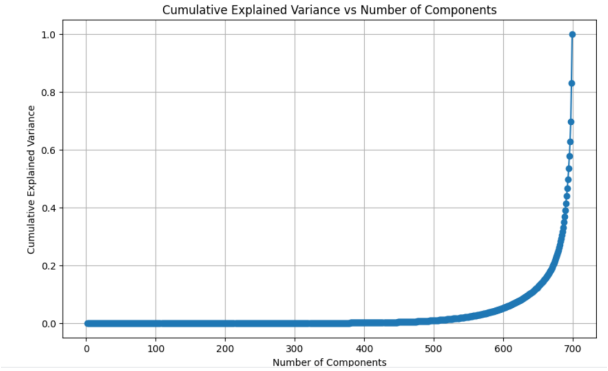

Figure 1: PCA analysis

PCA, we transformed the original feature space into a lower-dimensional one, capturing the most significant patterns in the data. Initially, we experimented with various numbers of principal components to balance between maintaining information and reducing computational overhead. After evaluating the explained variance across different components, we determined that the **optimal number of components was 100**. This choice preserved most of the variance in the data while significantly reducing the feature space, leading to faster training times and reduced memory usage.

6.2 Standardization of Features

Once the dimensionality was reduced, we applied a **Standard Scaler** to normalize the feature values. This step was critical because models such as Support Vector Regressors (SVR) and Gradient Boosting Regressors (GBR) are sensitive to the scale of the input features. Standardization ensures that all features contribute equally to the model and helps in faster convergence during training.

The Standard Scaler transformed the data to have a mean of 0 and a standard deviation of 1. This normalization step was applied both during training and evaluation to maintain consistency across the datasets.

6.3 Training and Evaluation

With the reduced feature space (100 components) and standardized data, we proceeded to train and evaluate the models. The use of PCA not only reduced the number of features but also accelerated the training process. Specifically: - **SVR**: The reduced number of dimensions allowed for quicker kernel computations, especially with the RBF kernel. This led to faster model training while maintaining competitive performance in terms of ranking quality. - **GBR**: With fewer features, the decision trees within the ensemble could be trained more efficiently, reducing the complexity of each tree and speeding up the overall model. - **MLP**: The reduced dimensionality lowered the number of weights and biases in the neural network, thus shortening the forward and backward propagation times during training.

In addition to speed improvements, the PCA transformation and scaling enhanced model generalization by removing noise and irrelevant features. This resulted in improved evaluation metrics such as nDCG, while the reduction in training time allowed for more extensive hyperparameter tuning, ultimately leading to better model performance.

Model	**nDCG@5**	**nDCG@10**	**nDCG@5**
SVR	0.66	0.72	0.81
GBP	0.64	0.70	0.81
MLP	0.71	0.75	0.84

Table 2: Performance comparison across different models

By leveraging PCA and standardization, we successfully addressed both speed and performance bottlenecks, ensuring that the models could handle larger datasets efficiently without compromising accuracy.