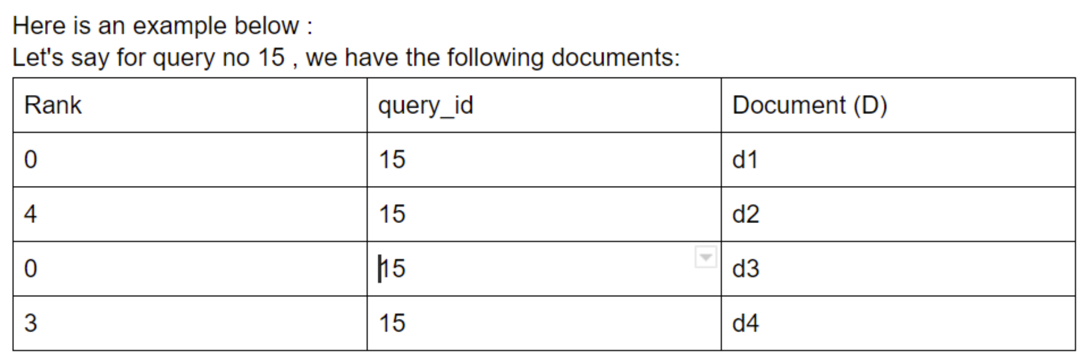
To rank and find out the best set of documents amongst all the other documents for a particular query search.

Motivation: Ranking creation is aimed at creating a ranking list of offerings based on the features of the offerings and the request so that ‘good’ offerings to the request are ranked at the top. Out of a large number of web pages available over the internet, it becomes cumbersome and time taking to search for the web page actually containing the required information. There are algorithms such as LambdaRank, RankNet, etc. which work over it using preference learning. We’ve tried to improve them and make them more efficient by applying some changes in the algorithms and the neural network used.

E**laboration of the model:** Given a query from the user, the system retrieves documents containing the query words from the collection, ranks the documents on the basis of the relevancy of the document for that particular query. Every feature of the document is assigned a weight with the aid of neural networks. The score of each document is then predicted on the basis of the weights using LambdaRank and RankNet. Finally, the document with the highest score is listed first and the rest follow.

Input/Training Data: About the dataset: We have used the Microsoft Web 10k dataset. Website Link for the Dataset: Microsoft Learning to Rank Official Website link for the dataset: Microsoft Learning to Rank Dataset The dataset consists of documents corresponding to each query. A document is ranked between 0 to 4 with 0 being the least relevant to the query and 4 being the most relevant to the query. Our task is to rank the dataset that correctly predicts how relevant a document is to a particular query. For this, we convert the dataset into pairwise data. The pairwise comparison is made between 2 documents of the same query and is not made between documents of different queries. Here is an example figure on the next page. The following pairwise data is formed. (d2,d1 (d3,d2) (d4,d3) (d3,d0) We don't take two documents with the same rank for training.

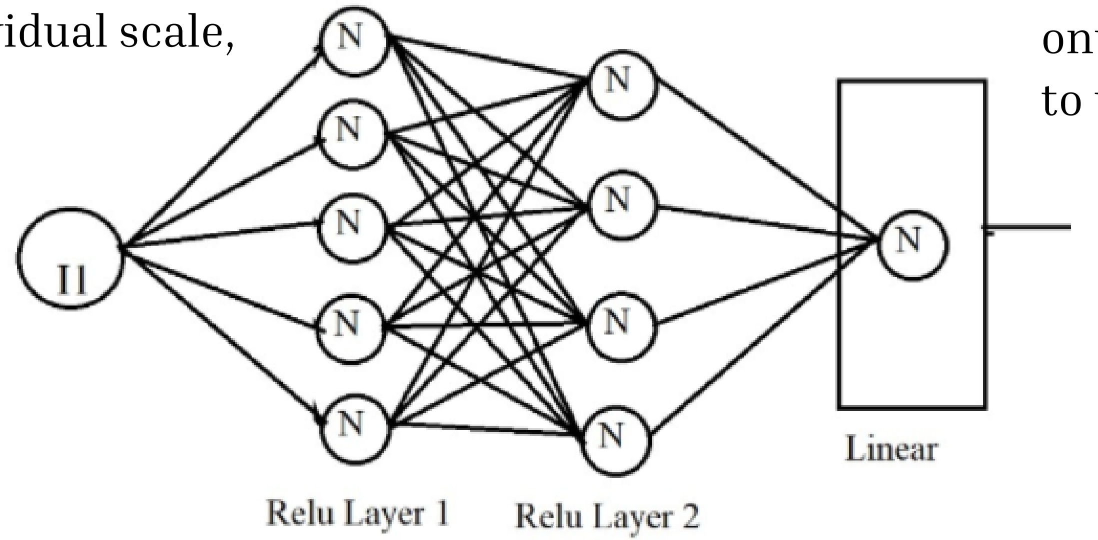
We also create an output node. Rather than just creating an output value of 0/1 which says which document is preferable, we decided to use a probabilistic function that tells us how important that document is on a comparison level.


Here is an example below :
Let's say for query no 15 , we have the following documents:

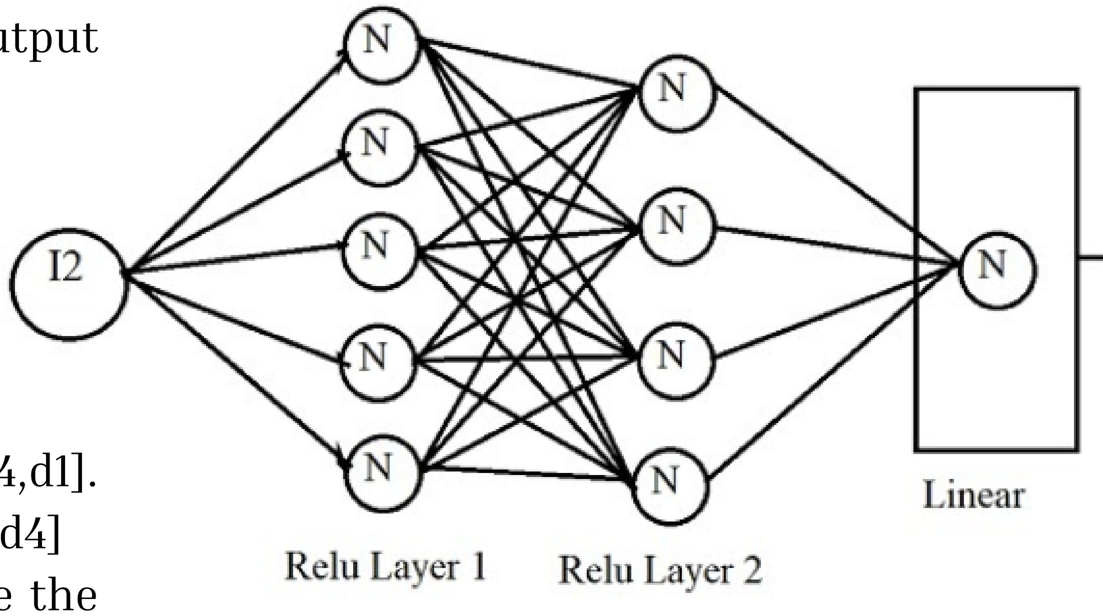
Rank	query_id	Document (D)
0	15	d1
4	15	d2
0	15	d3
3	15	d4

We calculate it using : P=exp(y2-y1)/(1+exp(y2-y1))
For documents d2 ,d1 where d2>d1. So our output corresponding to this pairwise data will be :
(d2,d1):0.982013
(d3,d2):0.017987
(d2,d4):0.7310585786300049
(d4,d3):0.9525741268224333
(d1,d4): 0.04742587317756746
In X1 array, we store the following : [d2,d3,d2,d4,d1].
In X2 array ,we store the following : [d1,d2,d4,d3,d4]
In the corresponding output array (y), we store the following [0.98,0.01,0.73,0.95,0.04].
So, our final transformed data would look like this : X1,X2,y.

Structure of Neural Network: Our model is a Neural Network that takes 2 input feature vectors First, at an individual scale,

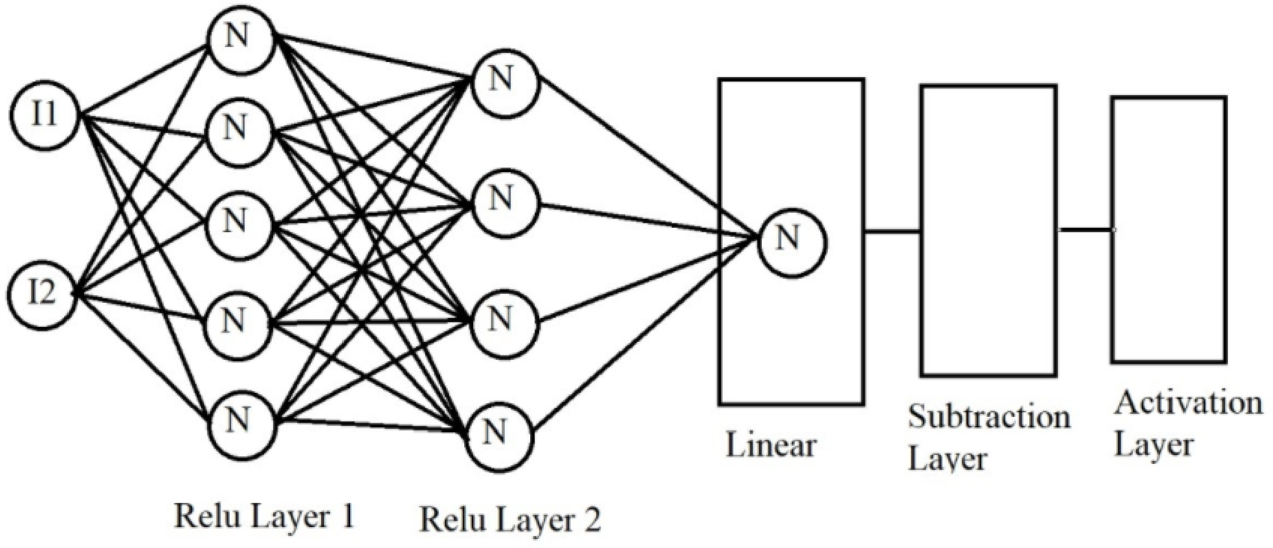


I1 takes the array X1 as input. It s passed onto layer 1 that uses Relu as its activation function. Then it is passed onto layer 2 that also uses Relu as its activation function. Then it is passed onto Layer 3 which uses a linear function to transform into a linear output.
Similarily,



I2 takes the array X2 as input.
It is passed onto layer 1 that uses Relu as its activation function. Then it is passed onto layer 2 that also uses Relu as its activation function. Then it is passed onto Layer 3 which uses a linear function to transform into a linear output.

As a whole, Now that we have the outputs from the Linear Layer from Input1 and Input 2, We subtract those values at the subtract layer and pass them onto the activation layer.



Finally, the value that we have at the activation is compared to which class it belongs to. The activation layer uses a sigmoid function that gives a value between 0 to 1. This value is compared with the original output value and the loss is calculated through binary cross-entropy. After this, we used the 'Adam Optimisation Algorithm' over the classical Stochastic Gradient Descent method to traverse back the nodes and adjust the network weights accordingly

How is our algorithm different from the existing ones ? RankNet uses Adam optimization instead of SGD. We use Relu as our activation layers instead of using sigmoid activation layers. The implementation that we carried out using the pairwise preference was not to classify whether a document is definitely ahead of another. It was rather to specify the model that documents A is preferred to document B with a score of x. In our other model, we performed some bitwise operations to enhance the values. We chose the positive index and the negative index. Calculated the log inverse values of them. Made a bitwise shift to amplify the values. Multiplied this with log_inv. We follow this for both positive and negative labels. We then multiply the difference with the IDCG and set it as our initial weights. Here we just classify them as class 0 or class 1 depending upon whether document A is preferred over document B.

Evaluation Approaches and accuracy measured: NDCG (Normalized Discounted Cumulative Gain). DCG(Discounted Cumulative Gain) The accuracy of our model is measured through NDCG which is specifically used for measuring how good our predictions are in a ranking problem. DCG is a metric commonly used to rate the output of learning to rank models. nDcg is a measure of the relative goodness of the output of the ranking * algorithm, and takes values between 0 and 1. 1 denoting that the algorithm has optimally ordered the pages for a query. 0 denoting that the pages have been reverse ordered. We implement ndcg per query.

$$DCG_n = \sum_{i=1}^{n} \frac{rel_i}{\log_2^{i+1}},$$

$$NDCG_n = \frac{DCG_n}{IDCG_n},$$

iDCG is basically calculating DCG for the true rank. For example f our model predicted the following: 2,3,3,1,2

We know that the actual prediction must be: 3,3,2,2,1

We can calculate the DCG and iDCG in the following method :

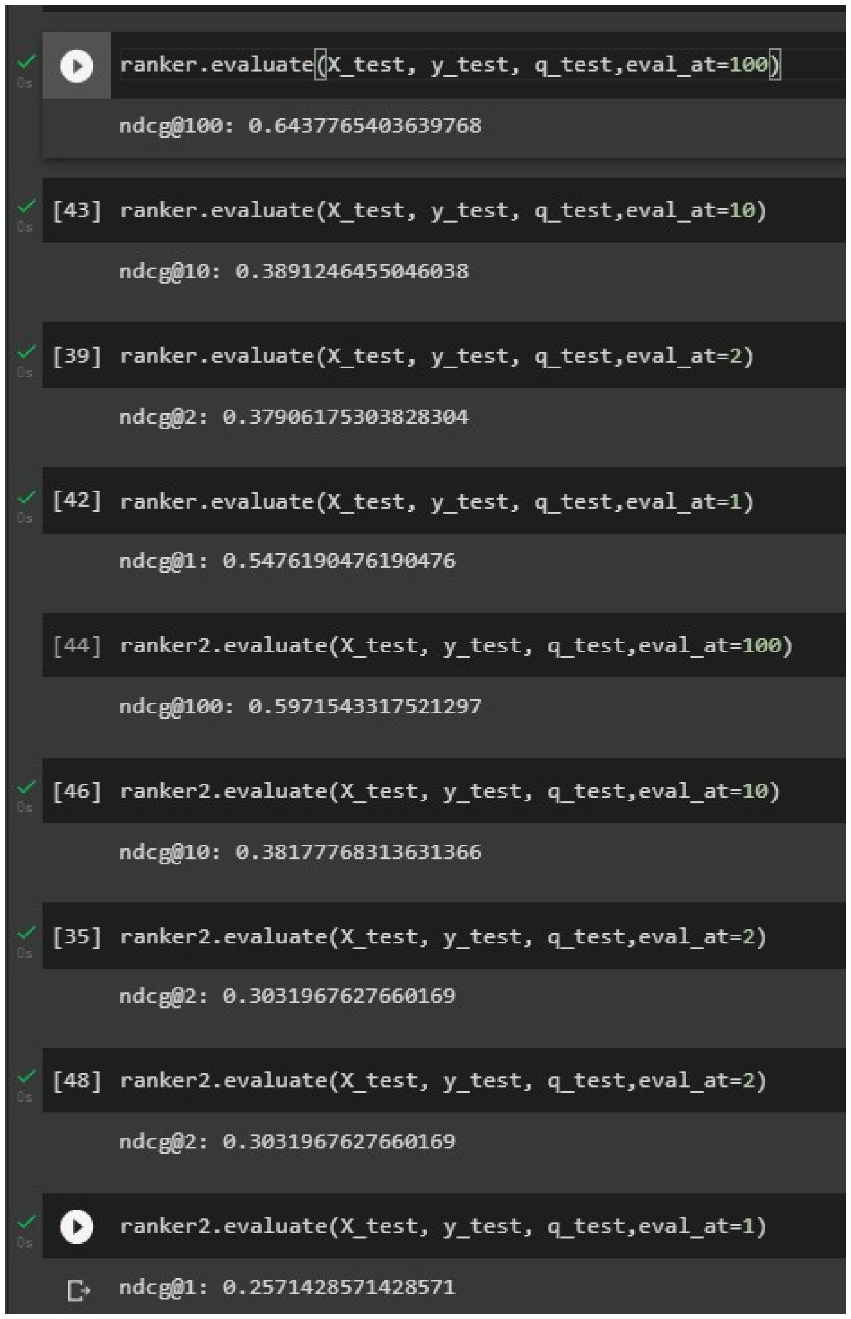
$$DCG = \frac{2}{log_2(1+1)} + \frac{3}{log_2(2+1)} + \frac{3}{log_2(3+1)} + \frac{1}{log_2(4+1)} + \frac{2}{log_2(5+1)} \approx 6.64$$

$$iDCG = \frac{3}{log_2(1+1)} + \frac{3}{log_2(2+1)} + \frac{2}{log_2(3+1)} + \frac{2}{log_2(4+1)} + \frac{1}{log_2(5+1)} \approx 7.14$$

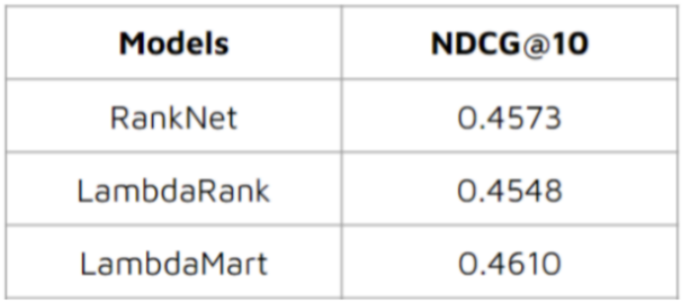
$$NDCG = \frac{DCG}{iDCG} = \frac{6.64}{7.14} \approx 0.93$$

Comparison with the baselines: The improvised model works with more accuracy as compared to the existing models such as LambdaRank and RankNet. The accuracy can be compared by taking NDCGs into consideration.

Our Model:
Ranker- LambdarRank
Ranker2- RankNet

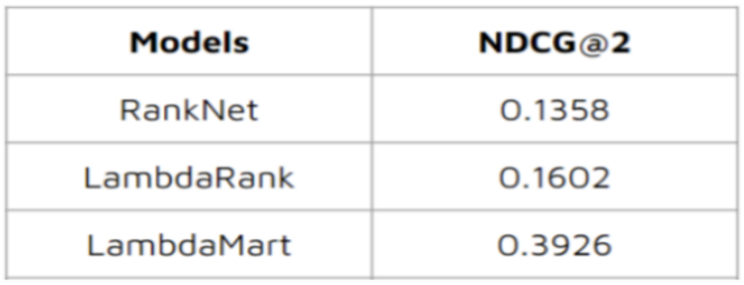


```
     ranker.evaluate(X_test, y_test, q_test,eval_at=100)
     ndcg@100: 0.6437765403639768

[43] ranker.evaluate(X_test, y_test, q_test,eval_at=10)
     ndcg@10: 0.3891246455046038

[39] ranker.evaluate(X_test, y_test, q_test,eval_at=2)
     ndcg@2: 0.37906175303828304

[42] ranker.evaluate(X_test, y_test, q_test,eval_at=1)
     ndcg@1: 0.5476190476190476

[44] ranker2.evaluate(X_test, y_test, q_test,eval_at=100)
     ndcg@100: 0.5971543317521297

[46] ranker2.evaluate(X_test, y_test, q_test,eval_at=10)
     ndcg@10: 0.38177768313631366

[35] ranker2.evaluate(X_test, y_test, q_test,eval_at=2)
     ndcg@2: 0.3031967627660169

[48] ranker2.evaluate(X_test, y_test, q_test,eval_at=2)
     ndcg@2: 0.3031967627660169

     ranker2.evaluate(X_test, y_test, q_test,eval_at=1)
     ndcg@1: 0.2571428571428571
```



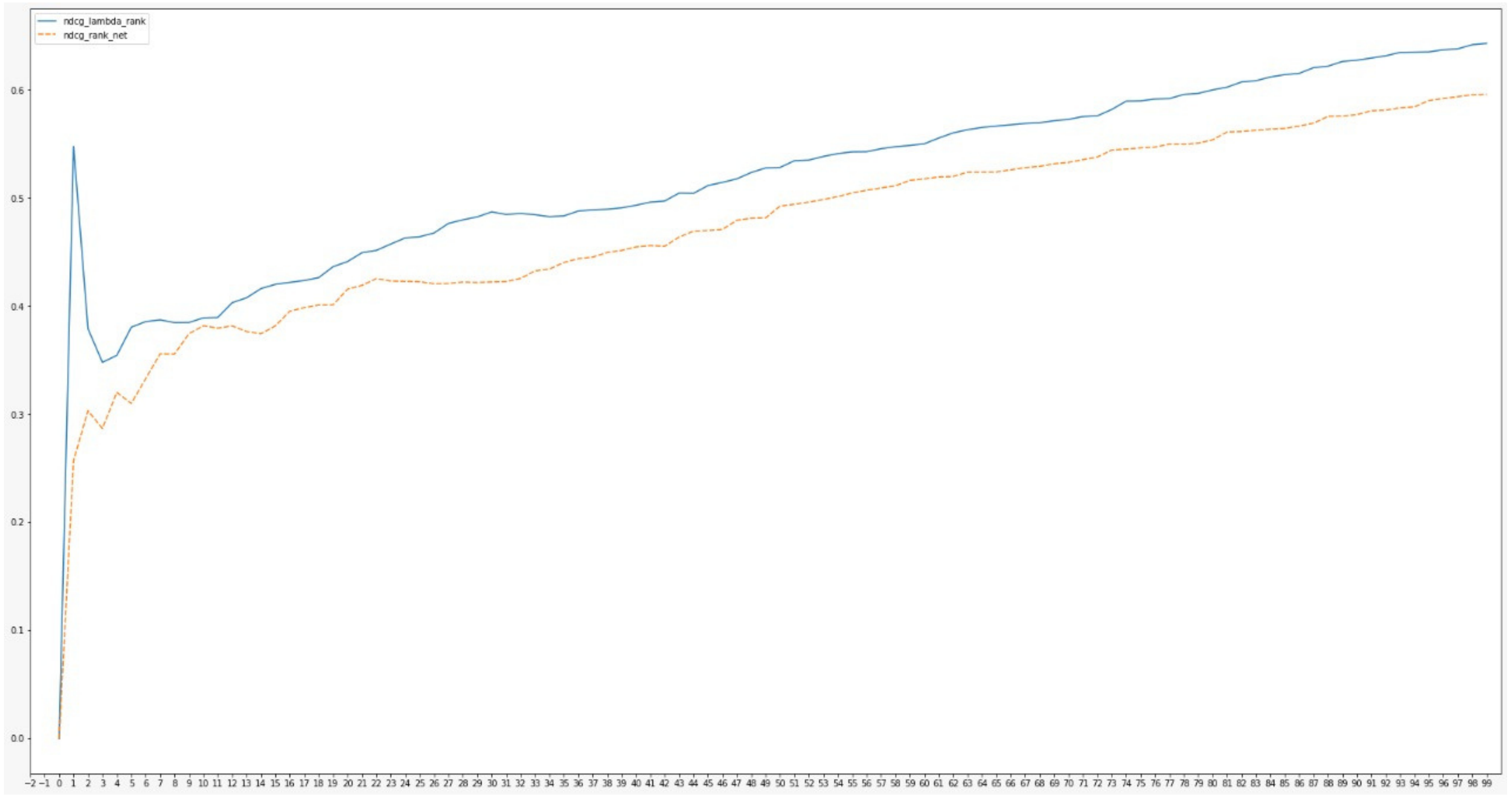
Models	NDCG@10
RankNet	0.4573
LambdaRank	0.4548
LambdaMart	0.4610

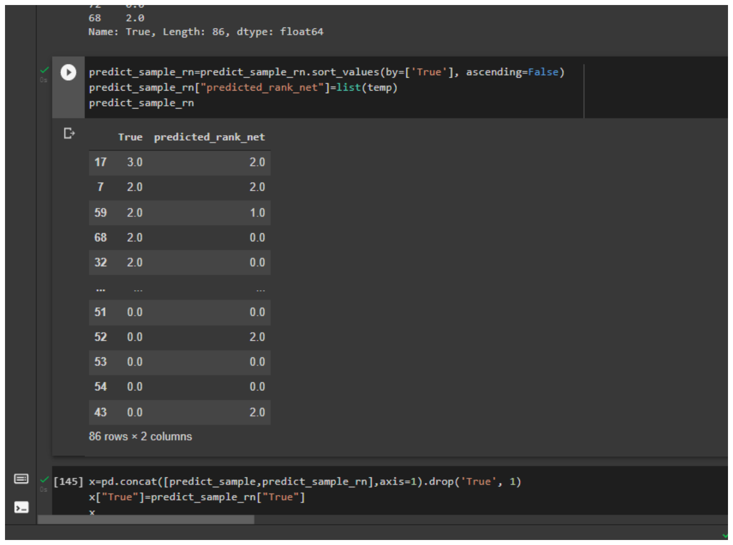
Figure 5: MQ2007 Dataset performance



Models	NDCG@2
RankNet	0.1358
LambdaRank	0.1602
LambdaMart	0.3926

Figure 6: MSLR Web-10K Dataset performance





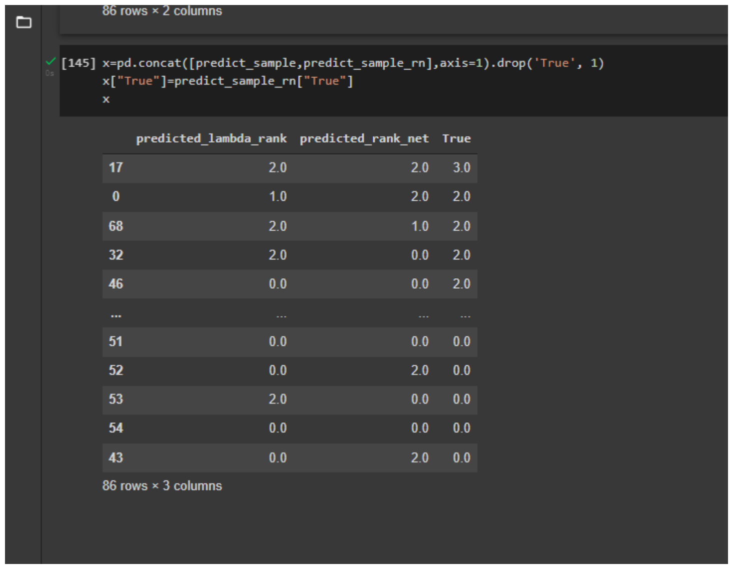
```
68    2.0
Name: True, Length: 86, dtype: float64

predict_sample_rn=predict_sample_rn.sort_values(by=['True'], ascending=False)
predict_sample_rn["predicted_rank_net"]=list(temp)
predict_sample_rn
```

	True	predicted_rank_net
17	3.0	2.0
7	2.0	2.0
59	2.0	1.0
68	2.0	0.0
32	2.0	0.0
...	...	...
51	0.0	0.0
52	0.0	2.0
53	0.0	0.0
54	0.0	0.0
43	0.0	2.0

86 rows × 2 columns

```
[145] x=pd.concat([predict_sample,predict_sample_rn],axis=1).drop('True', 1)
      x["True"]=predict_sample_rn["True"]
      x
```

The NDCGs obtained in the Existing models @2 and @10 are 0.1351 and 0.3565 respectively whereas the NDCG obtained by our model is close to .39 which shows that the improvised model shows more accuracy and is more efficient.

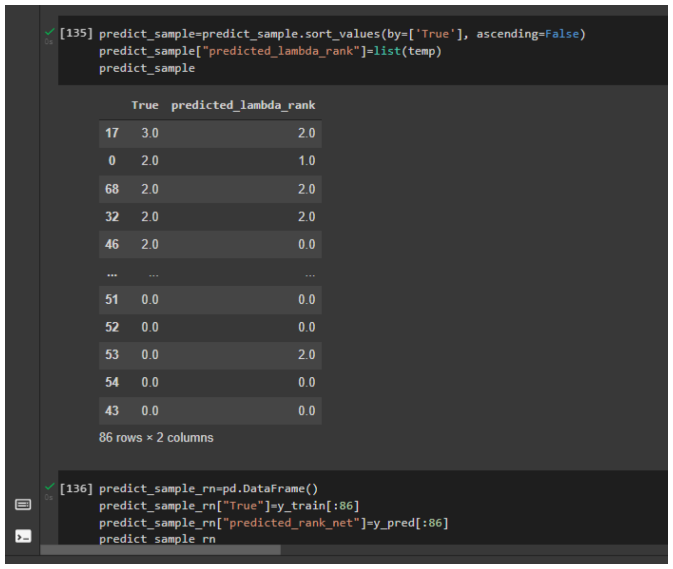


86 rows × 2 columns

```
[145] x=pd.concat([predict_sample,predict_sample_rn],axis=1).drop('True', 1)
      x["True"]=predict_sample_rn["True"]
      x
```

	predicted_lambda_rank	predicted_rank_net	True
17	2.0	2.0	3.0
0	1.0	2.0	2.0
68	2.0	1.0	2.0
32	2.0	0.0	2.0
46	0.0	0.0	2.0
...	...	...	...
51	0.0	0.0	0.0
52	0.0	2.0	0.0
53	2.0	0.0	0.0
54	0.0	0.0	0.0
43	0.0	2.0	0.0

86 rows × 3 columns



```
[135] predict_sample=predict_sample.sort_values(by=['True'], ascending=False)
      predict_sample["predicted_lambda_rank"]=list(temp)
      predict_sample
```

	True	predicted_lambda_rank
17	3.0	2.0
0	2.0	1.0
68	2.0	2.0
32	2.0	2.0
46	2.0	0.0
...	...	...
51	0.0	0.0
52	0.0	0.0
53	0.0	2.0
54	0.0	0.0
43	0.0	0.0

86 rows × 2 columns

```
[136] predict_sample_rn=pd.DataFrame()
      predict_sample_rn["True"]=y_train[:86]
      predict_sample_rn["predicted_rank_net"]=y_pred[:86]
      predict_sample_rn
```