Model Debugging and Testing

An introduction to model debugging and how it is conducted for a machine learning system.

We'll cover the following

Let's go over different phases in the development of a machine learning system, potential issues that we can face, and how to debug and fix them.

There are two main phases in terms of the development of a model that we will go over:

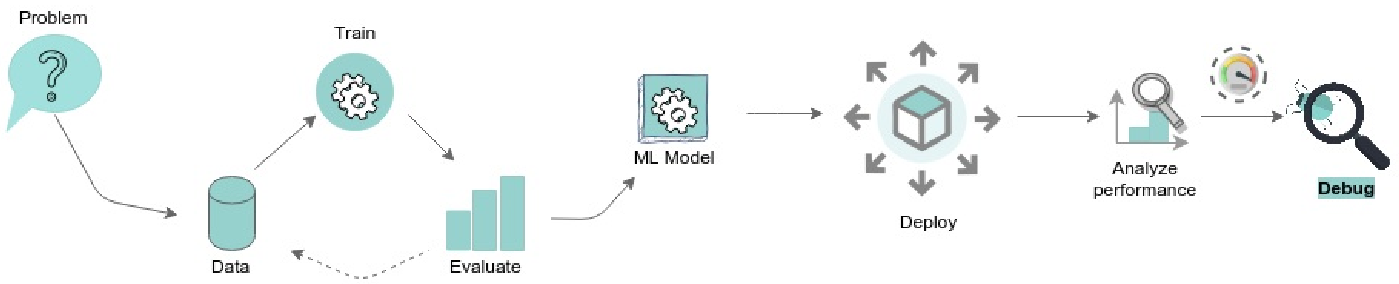
- Building the first version of the model and the ML system.
- Iterative improvements on top of the first version as well as debugging issues in large scale ML systems.

Building model v1

The goal in this phase is to build the 1st version of the model. Few important steps in this stage are:

- We begin by identifying a business problem in the first phase and mapping it to a machine learning problem.
- We then go onto explore the training data and machine learning techniques that will work best on this problem.
- Then we train the model given the available data and features, play around with hyper-parameters.
- Once the model has been set up and we have early offline metrics like accuracy, precision/recall, AUC, etc., we continue to play around with the various features and training data strategies to improve our offline metrics.
- If there is already a heuristics or rule-based system in place, our objective from the offline model would be to perform at least as good as the current system, e.g., for ads prediction problem, we would want our ML model AUC to be better than the current rule-based ads prediction based on only historical engagement rate.

It's important to get version 1 launched to the real system quickly rather than spending too much time trying to optimize it. For example, if our AUC is 0.7 and it's better than the current system with AUC 0.68, it's generally a better idea to take model online and then continue to iterate to improve the quality. The reason is primarily that model improvement is an iterative process and we want validation from real traffic and data along with offline validation. We will look at various ideas that can help in that iterative development in the following sections.



ML Model Workflow

Deploying and debugging v1 model

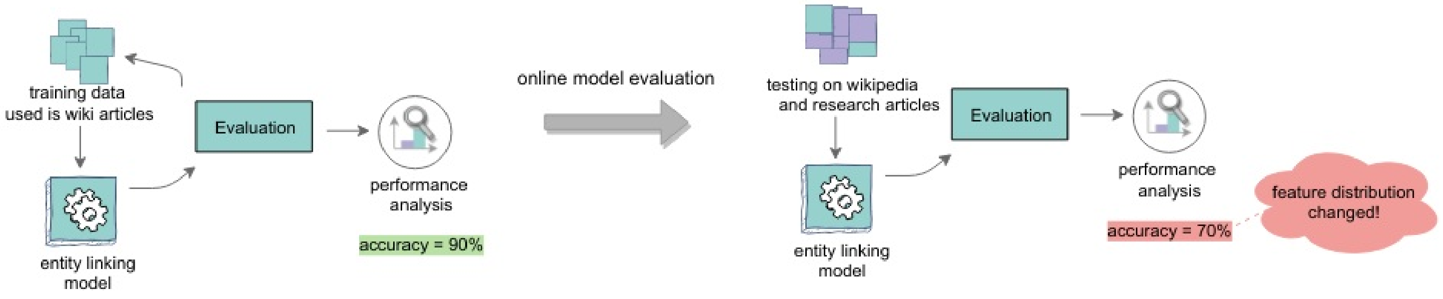
Some ML-based systems only operate in an offline setting, for example, detect objects from a large set of images. But, most systems have an online component as well, for example, building a search ranking ML model will have to run online to service incoming queries and ranking the documents that match the query.

In our first attempt to take the model online, i.e., enable live traffic, might not work as expected and results don't look as good as we anticipated offline. Let's look at a few failure scenarios that can happen at this stage and how to debug them.

Change in feature distribution

The change in the feature distribution of training and evaluation set can negatively affect the model performance. Let's consider an example of an Entity linking system that is trained using a readily available Wikipedia dataset. As we start using the system for real traffic, the traffic that we are now getting for finding entities is a mix of Wikipedia articles as well as research papers. Given the model wasn't trained on that data, its feature distribution would be a lot different than what it was trained on. Hence it is not performing as well on the research articles entity detection.

Another scenario could be a significant change in incoming traffic because of seasonality. Let's consider an example of a search system trained using data for the last 2 weeks of December, i.e., mostly holiday traffic. If we deploy this system in January, the queries that it will see will be vastly different than what it was trained on and hence not performing as well as we observed in our offline validation.

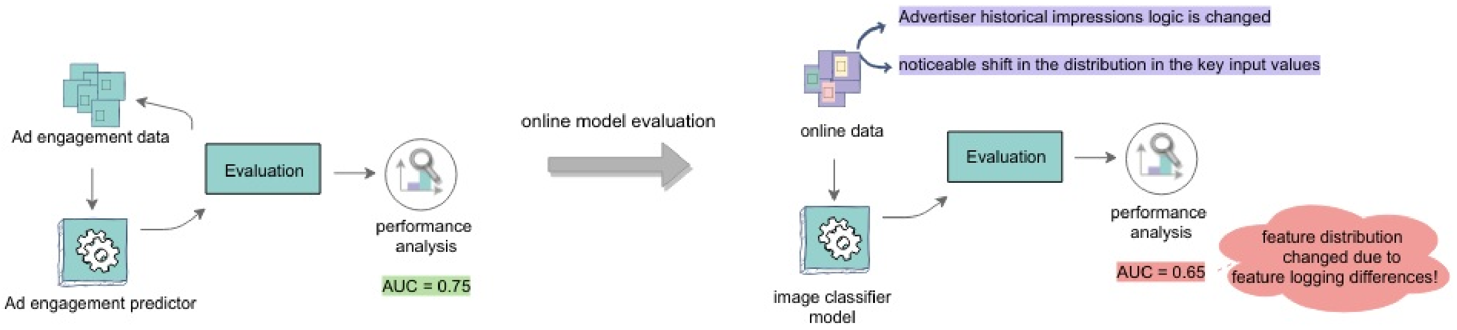


Entity linking model could not work online due to change in feature distribution

Feature logging issues

When the model is trained offline, there is an assumption that features of the model generated offline would exactly be the same when the model is taken online. However, this might not be true as the way we generated features for our online system might not exactly be the same. It's a common practice to append features offline to our training data for offline training and then add them later to the online model serving part. So, if the model doesn't perform as well as we anticipated online, it would be good to see if feature generation logic is the same for offline training as well as online serving part of model evaluation.

Suppose we build an ads click prediction model. The model is trained on historical ad engagement. We then deploy it to predict the ads engagement rate. Now the assumption is that the features generated for the model offline would exactly be the same for run-time evaluation. Let's assume that we have one important feature/signal for our model that's based on historical advertiser ad impressions. During training, we compute this feature by using the last 7 days' impression. But, the logic to compute this feature at model evaluation time uses the last 30 days of data to compute advertiser impressions. Because of this feature computed differently at training time and evaluation time, it will result in the model not performing well during online serving. It would be worth comparing the features used for training and evaluation to see if there is any such discrepancy.



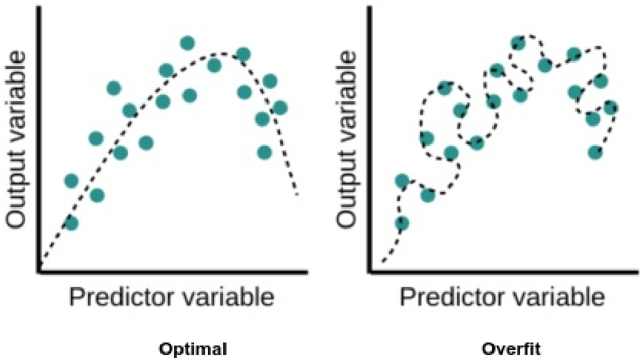
Ads prediction online metrics drop due to feature computation differences

Overfitting

Overfitting happens when a model learns the intrinsic details in the training data to the extent that it negatively impacts the performance of the model on new or unseen data.

If our model performance is lower in the live system but it still performs well on our training and validation set then there is a good chance that we have overfit our data by trying to improve the performance a bit too much. One good way of ensuring that we don't get into this problem is to use a hidden test set which is not used for tuning hyperparameters and only use it for final model quality measurement.

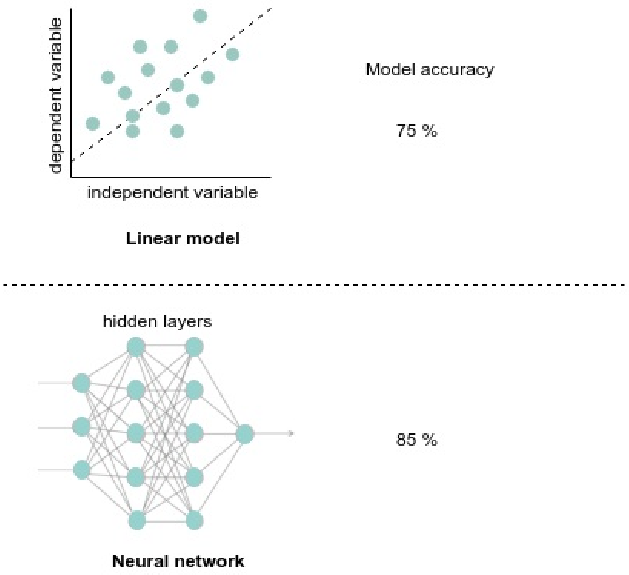
Another important part is to have a comprehensive and large test set to cover all possible scenarios in a fairly similar distribution to how we anticipate them in live traffic. For example, consider an image object prediction system whose test set only has large-sized objects - (covering 50% pixels or more of the image) for 90% samples and small-sized objects for 10% samples (covering 10% pixels or less). If the live traffic has the opposite distribution of large and small size objects, then the model might not perform well on the live set.



Overfit and optimal model

Under-fitting

One indication from training the model could be that the model is unable to learn complex feature interactions especially if we are using a simplistic model. So, this might indicate to us that using slightly higher-order features, introduce more feature interactions, or use a more complex /expensive model such as a neural network.



Neural network outperforms linear regression model because it can capture non-linear relationship between features

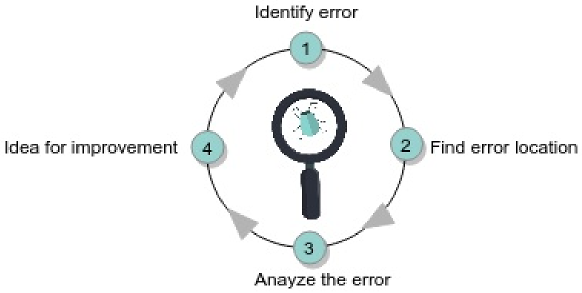
Iterative model improvement

As discussed above that the first version of the model deployed is generally not the most optimized one as our goal is to get the ML system out sooner to get feedback. Afterward, *Model debugging and testing* is generally done to come up with ideas to improve the quality of the model resulting in our metrics improvement.

Few cases that we discussed above regarding overfitting and under-fitting are still the questions that we should continue to ask during iterative model improvement but let's discuss a few more below.

The best way to iterative improve model quality is to start looking at failure cases of our model prediction and using that come up with the ideas that will help in improving model performance in those cases.

Let's go over some common types of ideas that will emerge as part of these failures examples debugging process.



Debugging flow

> Debugging in current context doesn’t mean to optimize overall architecture and system design that we have deeply discussed in this course like how to set up a problem, setting up architecture etc. It comprises of general methods that we use to observe issues in current model to continue to optimize it.

Missing important feature

Digging deeper into failures examples can identify missing features that can help us perform better in failures cases, e.g., consider a scenario where a movie actually liked by the user was ranked very low by our recommendation system. On debugging, we figure out that the user has previously watched two movies by the same actor, so adding a feature on previous ratings by the user for this movie actor can help our model perform better in this case.

Insufficient training examples

We may also find that we are lacking training examples in cases where the model isn’t performing well. We will cater to all possible scenarios where the model is not performing well and update the training data accordingly. For example, for the image segmentation problem, the segmentation model is not able to segment the image when there are multiple traffic lights. One way to look at that point will be to count the number of such training examples that we have in our data set with multiple traffic lights.

So we will add more training examples of multiple traffic lights and enable the model to learn to segment multiple traffic lights better. The following illustration shows the above concept by taking an example of a test image from cityscapes dataset.

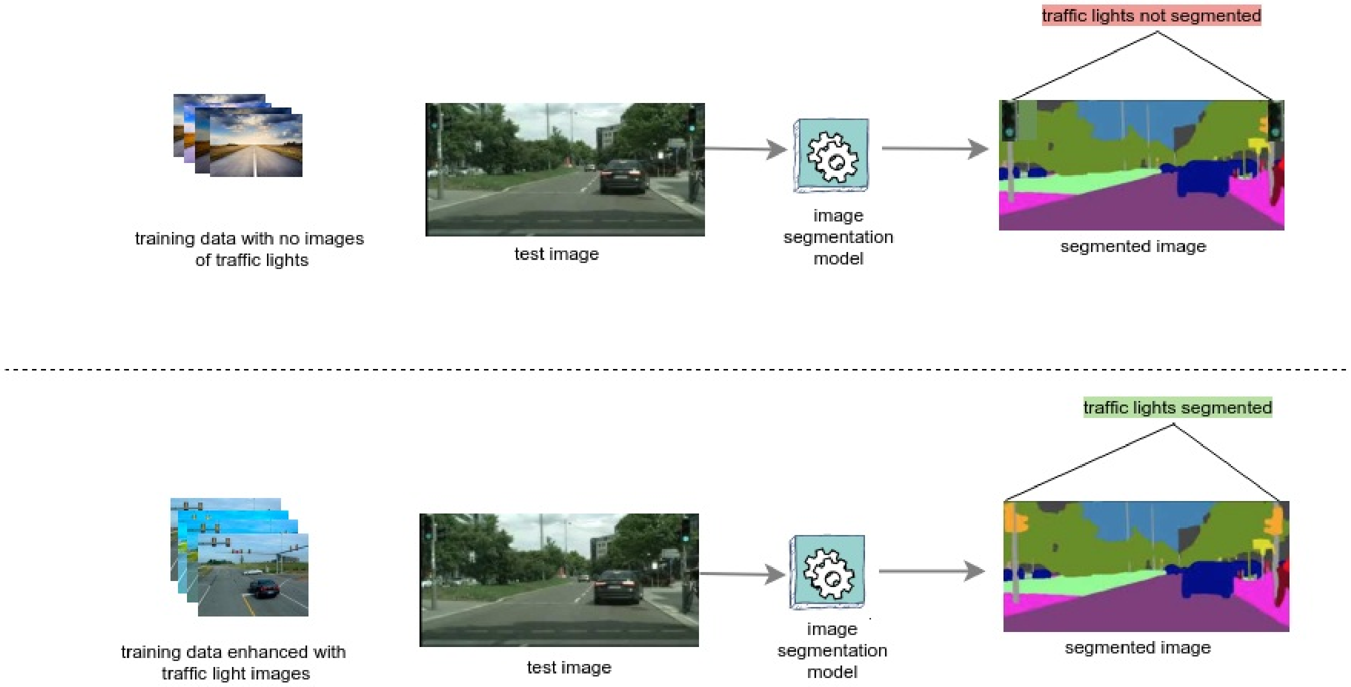

traffic lights not segmented
training data with no images of traffic lights
test image
image segmentation model
segmented image
traffic lights segmented
training data enhanced with traffic light images
test image
image segmentation model
segmented image

image segmentation model output performance increased in case of sufficient training data

Debugging large scale systems

In the case of debugging large scale systems with multiple components(or models), we need to see which part of the overall system is not working correctly. It could be done for one failure example or over a set of examples to see where the opportunity lies to improve the metrics.

The following are a few key steps to think about iterative model improvement for large scale end to end ML systems:

- **Identify the component**

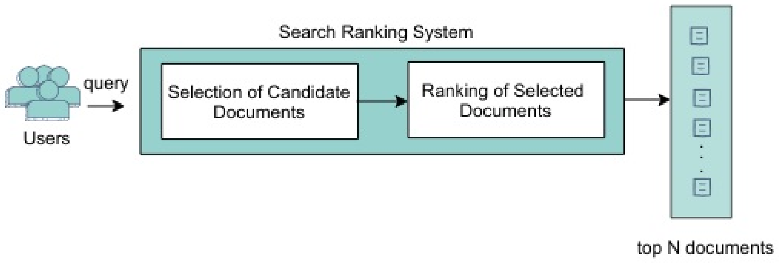
  This accounts for finding the architectural component resulting in a high number of failures in our failure set. Suppose that in the case of the search ranking system, we have opted for a layered model approach that we will discuss in detail in our Search problem discussion.

  In order to see the cause of failure, we will look at each layers' performance to understand the opportunity to significantly improve the quality of our search system. Let's assume that our search system has two key components 1) *Document selection* 2) *Ranking* of selected documents. Document selection focus is to ensure that all the top relevant documents get selected for the query while Ranker then ensures that our rank order is correct based on the relevance of the top 100 documents.

  If we look at few hundred failures of our overall search system, we should be able to identify the component that's resulting in more failures and as a result, decide to improve the quality of that component, e.g., if 80% of our overall search system failures are because of the ideal document not being selected in candidate selection component, we will debug the model deeply in that layer to see how to improve the quality of that model. Similarly, if ideal documents are mostly selected but are ranked lower by our ranking component, we will invest in finding the reason for failures in that layer and improve the quality of our ranking model.

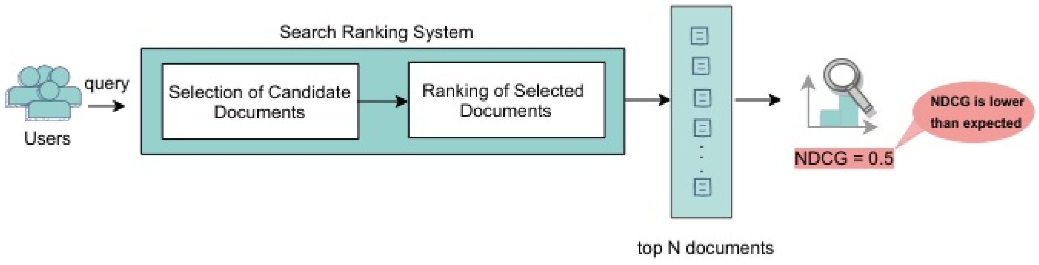
- **Improve the quality of component**

  Some of the model improvement methods that we have discussed above like adding more training data, features, modeling approach in case of overfitting and underfitting will still be the same once we identify the component that needs work, e.g., if we identify that the candidate selection layer needs improvement in our search, we will try to see missing features, add more training data or play around with ML model parameters or try a new model.


Search Ranking System
Users
query
Selection of Candidate Documents
Ranking of Selected Documents
top N documents

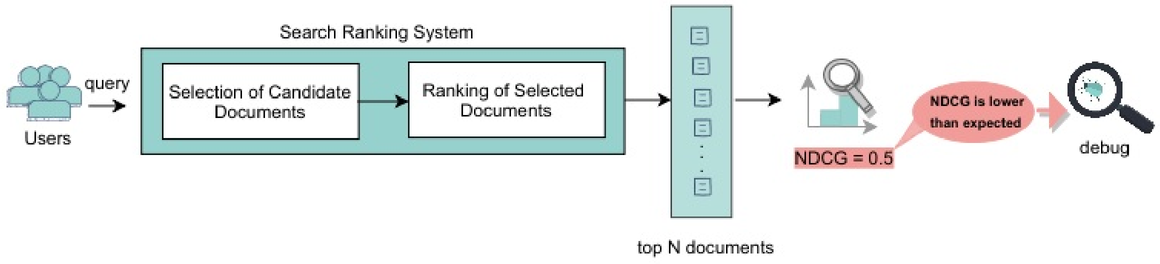
Search ranking system

1 of 10


Search Ranking System
Users
query
Selection of Candidate Documents
Ranking of Selected Documents
top N documents
NDCG = 0.5
NDCG is lower than expected

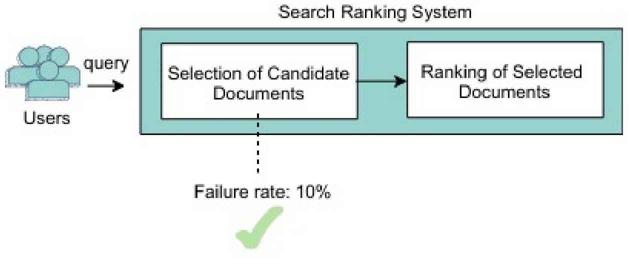
Evaluation of the system in terms of NDCG score

2 of 10


Search Ranking System
Users
query
Selection of Candidate Documents
Ranking of Selected Documents
top N documents
NDCG = 0.5
NDCG is lower than expected
debug

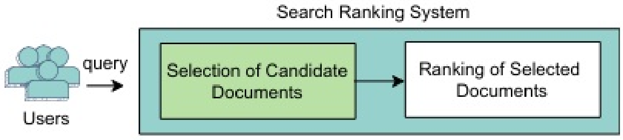
Evaluation of the system shows NDCG score is very low. So debugging is required.

3 of 10


Search Ranking System
Users
query
Selection of Candidate Documents
Ranking of Selected Documents
Failure rate: 10%

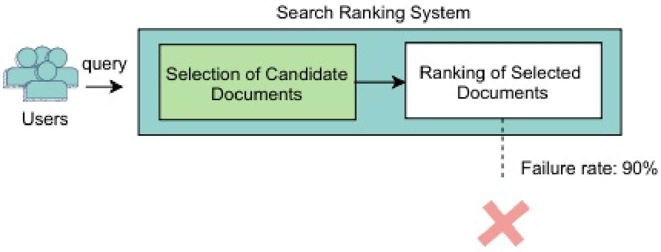
Check failure rate of Document Selection component

4 of 10


Search Ranking System
Users
query
Selection of Candidate Documents
Ranking of Selected Documents

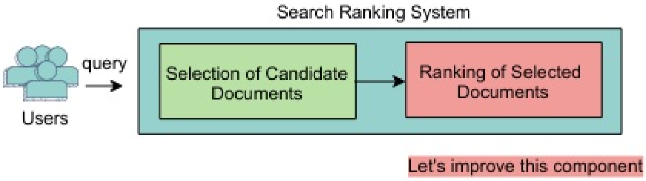
The selection component is working well

5 of 10


Search Ranking System
Users
query
Selection of Candidate Documents
Ranking of Selected Documents
Failure rate: 90%

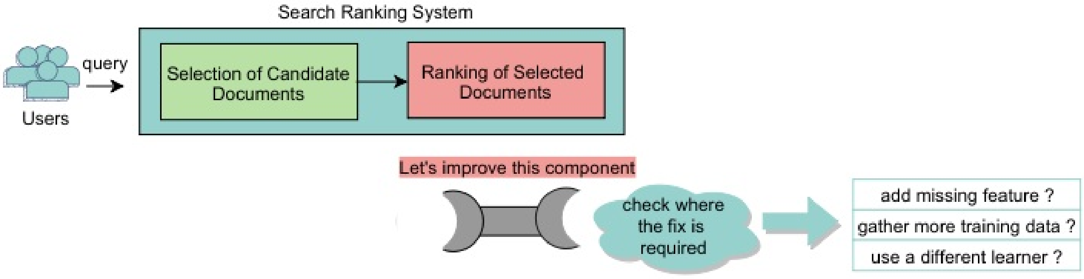
Failure rate is high for ranking component while debugging our failure set

6 of 10


Search Ranking System
Users
query
Selection of Candidate Documents
Ranking of Selected Documents
Let's improve this component

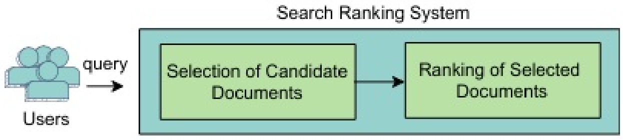
Improve ranking component ML model

7 of 10


Search Ranking System
Users
query
Selection of Candidate Documents
Ranking of Selected Documents
Let's improve this component
check where the fix is required
add missing feature ?
gather more training data ?
use a different learner ?

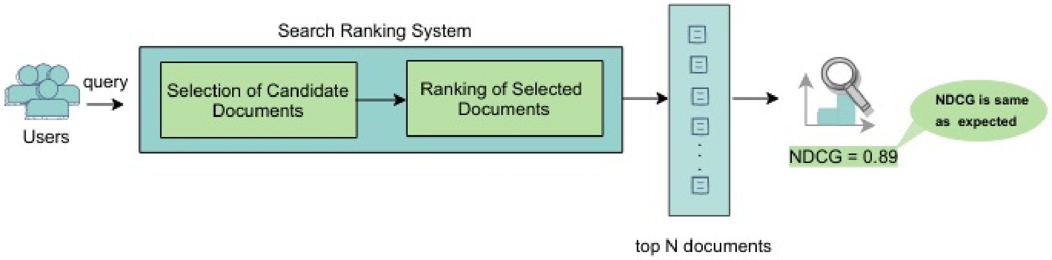
Improve model in ranking by model improvement strategies with features, training data or ML model

8 of 10


Search Ranking System
Users
query
Selection of Candidate Documents
Ranking of Selected Documents

The ranking component is fixed now

9 of 10



NDCG of the system is improved after bug fixing

10 of 10

← Back

Transfer Learning

Next →

Problem Statement

☑ Mark as Completed

Report an Issue Ask a Question