# HW2

**Name**: Haoyang Ling
**Email**: hyfrankl@umich.edu

### Q1

In all there are 118 categories.

```
Category:All articles lacking in-text citations
Category:All articles with specifically marked weasel-worded phrases
Category:Articles containing Dutch-language text
Category:All articles with vague or ambiguous time
Category:CS1: long volume value
Category:CS1 Spanish-language sources (es)
Category:Articles containing Greek-language text
Category:Articles containing traditional Chinese-language text
Category:CS1 Russian-language sources (ru)
Category:All pages needing factual verification
Category:All Wikipedia articles written in Australian English
Category:All Wikipedia articles needing clarification
Category:All articles with style issues
Category:All articles covered by WikiProject Wikify
Category:20th-century American male actors
Category:All BLP articles lacking sources
Category:CS1: Julian–Gregorian uncertainty
Category:All articles with failed verification
Category:All articles with bare URLs for citations
Category:Articles with bare URLs for citations from March 2022
Category:All articles to be expanded
Category:Short description is different from Wikidata
Category:Articles containing Chinese-language text
Category:Articles with FAST identifiers
Category:Articles with CINII identifiers
Category:Articles using infobox ethnic group with image parameters
Category:Articles containing Italian-language text
Category:All articles with dead external links
Category:Articles with MusicBrainz place identifiers
Category:All accuracy disputes
Category:Coordinates on Wikidata
Category:Articles with MusicBrainz work identifiers
Category:Articles containing French-language text
Category:Articles with German-language sources (de)
Category:All articles with a promotional tone
Category:Articles with 'species' microformats
Category:Articles with Spanish-language sources (es)
Category:CS1 Dutch-language sources (nl)
Category:Commons category link is on Wikidata
Category:Articles containing video clips
Category:Articles with OS grid coordinates
Category:All articles lacking reliable references
Category:Short description with empty Wikidata description
Category:Articles with LNB identifiers
Category:CS1 German-language sources (de)
Category:All articles needing coordinates
Category:Articles using infobox television channel
Category:Articles with Curlie links
Category:Articles with CANTICN identifiers
Category:Articles containing German-language text
Category:Articles containing Ancient Greek (to 1453)-language text
Category:Articles with WorldCat identifiers
Category:Official website different in Wikidata and Wikipedia
Category:Articles containing Russian-language text
Category:Articles with ISNI identifiers
Category:Articles with short description
Category:Articles with VIAF identifiers
Category:Articles containing explicitly cited English-language text
Category:Articles containing Latin-language text
Category:Articles with BNE identifiers
Category:All articles with unsourced statements
Category:All articles that may contain original research
Category:Articles with PDF format bare URLs for citations
Category:Articles using NRISref without a reference number
Category:Articles containing Japanese-language text
Category:CS1 maint: bot: original URL status unknown
```

```
Category:Articles with French-language sources (fr)
Category:CS1 French-language sources (fr)
Category:Articles with SUDOC identifiers
Category:All Wikipedia articles in need of updating
Category:Articles using infobox university
Category:All Wikipedia neutral point of view disputes
Category:Articles with hAudio microformats
Category:"Related ethnic groups" needing confirmation
Category:Webarchive template wayback links
Category:CS1 maint: url-status
Category:Articles with NARA identifiers
Category:Articles with MusicBrainz area identifiers
Category:Short description matches Wikidata
Category:Articles with hCards
Category:All Wikipedia articles written in American English
Category:Articles using small message boxes
Category:All articles containing potentially dated statements
Category:All articles with incomplete citations
Category:Articles with permanently dead external links
Category:Good articles
Category:All pages needing cleanup
Category:All stub articles
Category:CS1 maint: multiple names: authors list
Category:CS1 maint: archived copy as title
Category:Articles with LCCN identifiers
Category:Articles with BIBSYS identifiers
Category:Commons category link from Wikidata
Category:CS1 Japanese-language sources (ja)
Category:CS1 Italian-language sources (it)
Category:Articles with GND identifiers
Category:All articles needing additional references
Category:Articles with NKC identifiers
Category:Articles with SNAC-ID identifiers
Category:Articles with BNFdata identifiers
Category:Articles containing Korean-language text
Category:CS1 errors: missing periodical
Category:CS1 maint: unfit URL
Category:Articles containing Spanish-language text
Category:20th-century American male writers
Category:Articles containing Hebrew-language text
Category:Articles with MusicBrainz identifiers
Category:Articles with DTBIO identifiers
Category:CS1 errors: generic name
Category:Articles containing Portuguese-language text
Category:Articles with NDL identifiers
Category:All Wikipedia articles written in Indian English
Category:Articles with BNF identifiers
Category:Articles containing Arabic-language text
Category:Articles with J9U identifiers
Category:Articles containing simplified Chinese-language text
Category:All Wikipedia articles written in Canadian English
Category:Articles with multiple maintenance issues
```

### Q2

I add a new feature as the weighted unigram where I fetch the unigram frequency in documemnt that also appears in the query.

$$\text{score} = \frac{1}{|q|} \sum_{w_i \in q \cap d} c_d(w_i) \cdot \log(i+1)$$

, where $c_d(w_i)$ is in the descending order and q is the set of query words. In this score, it add more weights to those with low unigram frequency so that it scores higher for those documents with high occurences and tends to prefer a more uniform word distribution. Noticeably, the highest N-gram frequency won't exceed the value of $c_d(w_N)$. So, this weighted score also servers an estimation of the N-grams frequency here. Plus, it will also normalize the score by considering the query length into account so that it can serve as a more robust feature for query and document interaction.

### Q3

- overall

| | pagerank | authority_score | hub_score |
|---|---|---|---|
| **0** | 23538754 | 23538754 | 18603746 |
| **1** | 19344515 | 14532 | 924166 |
| **2** | 31717 | 31717 | 13425800 |
| **3** | 14533 | 48361 | 277226 |
| **4** | 48361 | 26667 | 25954090 |

| | pagerank | authority_score | hub_score |
|---|---|---|---|
| **5** | 645042 | 53207 | 145422 |
| **6** | 30680 | 11867 | 40250996 |
| **7** | 5843419 | 5843419 | 307608 |
| **8** | 3434750 | 147101 | 5843419 |
| **9** | 32927 | 3434750 | 24296062 |

- pageranks

| | docid | title | pagerank |
|---|---|---|---|
| **0** | 23538754 | Wayback Machine | 0.002085 |
| **1** | 48361 | Geographic coordinate system | 0.001292 |
| **2** | 3434750 | United States | 0.001066 |
| **3** | 30680 | The New York Times | 0.000776 |
| **4** | 32927 | World War II | 0.000575 |
| **5** | 31717 | United Kingdom | 0.000572 |
| **6** | 645042 | New York City | 0.000467 |
| **7** | 14533 | India | 0.000465 |
| **8** | 5843419 | France | 0.000459 |
| **9** | 19344515 | The Guardian | 0.000445 |

- authority score

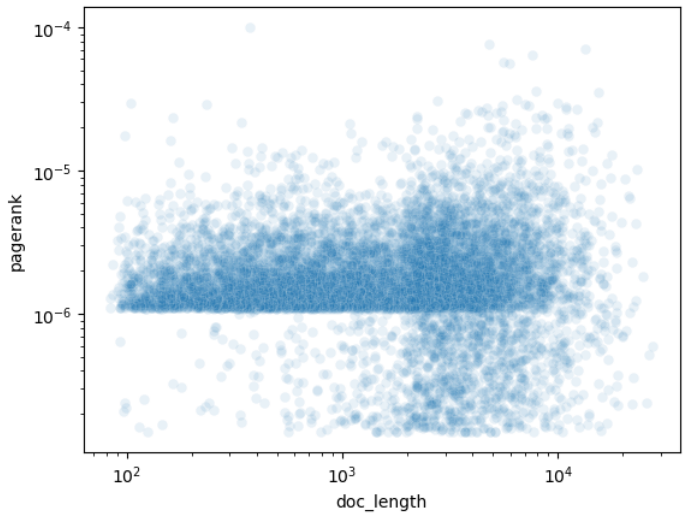
| | docid | title | authority_score |
|---|---|---|---|
| **0** | 23538754 | Wayback Machine | 0.297455 |
| **1** | 3434750 | United States | 0.244132 |
| **2** | 5843419 | France | 0.161043 |
| **3** | 31717 | United Kingdom | 0.150814 |
| **4** | 11867 | Germany | 0.147175 |
| **5** | 14532 | Italy | 0.133701 |
| **6** | 147101 | Record label | 0.132971 |
| **7** | 48361 | Geographic coordinate system | 0.130685 |
| **8** | 53207 | Record producer | 0.125643 |
| **9** | 26667 | Spain | 0.123123 |

- hub score

| | docid | title | hub_score |
|---|---|---|---|
| **0** | 13425800 | War on terror | 0.015191 |
| **1** | 18603746 | Beijing | 0.014729 |
| **2** | 277226 | Demonym | 0.014656 |
| **3** | 145422 | Joe Biden | 0.014571 |
| **4** | 24296062 | Foreign relations of the United States | 0.014549 |
| **5** | 307608 | Foreign relations of Canada | 0.014206 |
| **6** | 5843419 | France | 0.014082 |
| **7** | 924166 | LGBT rights by country or territory | 0.013867 |
| **8** | 25954090 | Visa requirements for United States citizens | 0.013744 |

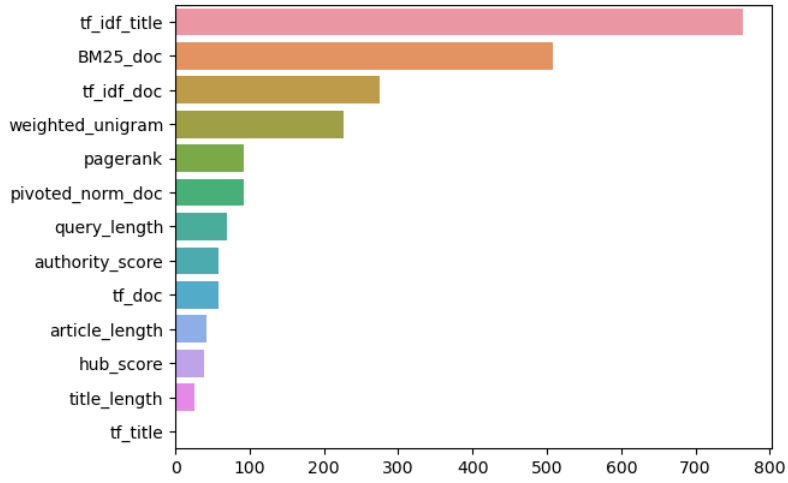
- Observations: many country names appear in this top lists. However, it is surprising to see that wayback machine gets high values in page rank and authority scores. However, the page rank identifies some newspaper like The New York Times and The Guardian. It is interesting to see that the hub score weighs higher on the foreign relation and legal status/citizenship.
- Most useful: I would like to define the good document with its importance/relevance in the whole websites. Page rank may serve as a good indicator because it will recommend The New York Times and The Guardian which are worldwide well-known newspaper websites.

### Q4




- I draw it in the loglog space and find that there is a distinct boundary when pagerank is around $10^{-6}$. And, the high document length tends to lead high page rank with high variance because the more content the document has, the more likely it will be to interact with other documents.

## Q5



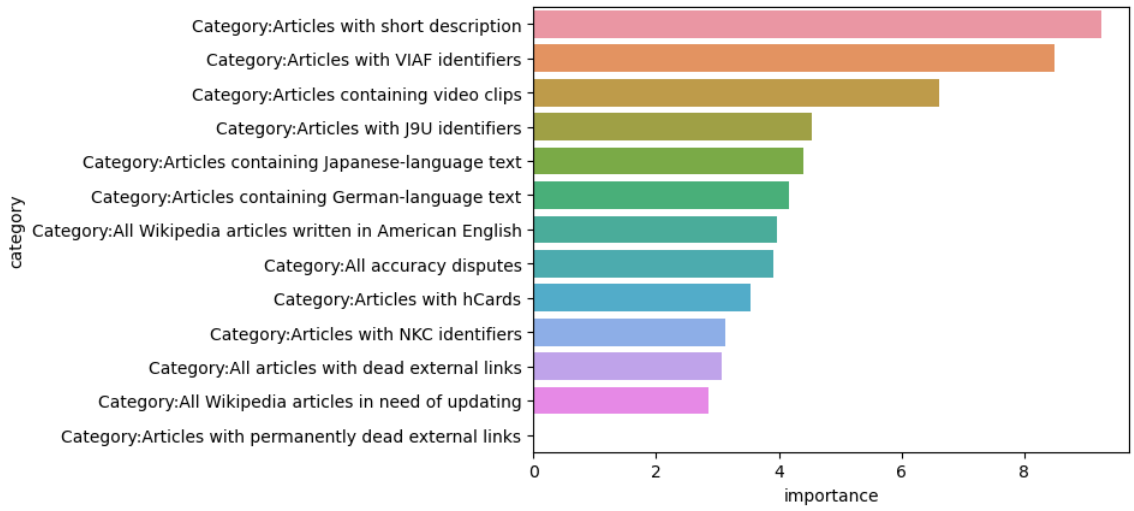

- In the picture, we can find that BM25 and TF-IDF do a good job. And, the added feature `weighted_unigram` also performs well. However, it is astonishing to find that `tf-idf-title` is the dominant factor that solely focusing on the title can do a good job.

| | feature | importance |
|---|---|---|
| **0** | tf_idf_title | 764.455770 |
| **1** | BM25_doc | 507.411774 |
| **2** | tf_idf_doc | 274.577948 |
| **3** | weighted_unigram | 226.078532 |
| **4** | pagerank | 92.240010 |
| **5** | pivoted_norm_doc | 92.201091 |
| **6** | query_length | 70.105271 |
| **7** | authority_score | 57.673680 |
| **8** | tf_doc | 57.404420 |
| **9** | article_length | 42.369930 |
| **10** | hub_score | 38.622510 |
| **11** | title_length | 25.506510 |
| **12** | tf_title | 0.000000 |

## Q6

| | category | importance |
|---|---|---|
| **25** | Category:Articles with short description | 9.24552 |
| **24** | Category:Articles with VIAF identifiers | 8.49493 |
| **35** | Category:Articles containing video clips | 6.61918 |

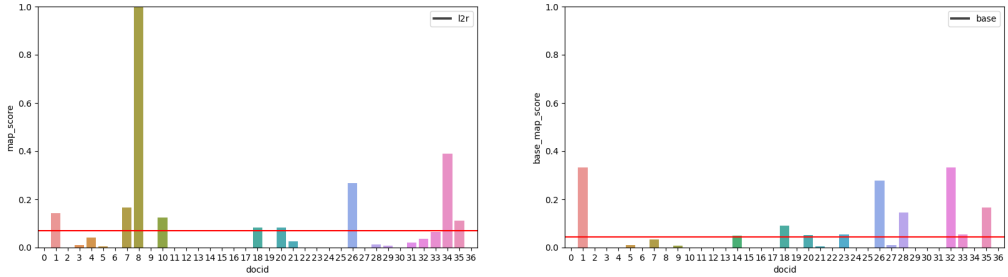
| | category | importance |
|---|---|---|
| **16** | Category:Articles with J9U identifiers | 4.53467 |
| **103** | Category:Articles containing Japanese-language... | 4.39246 |



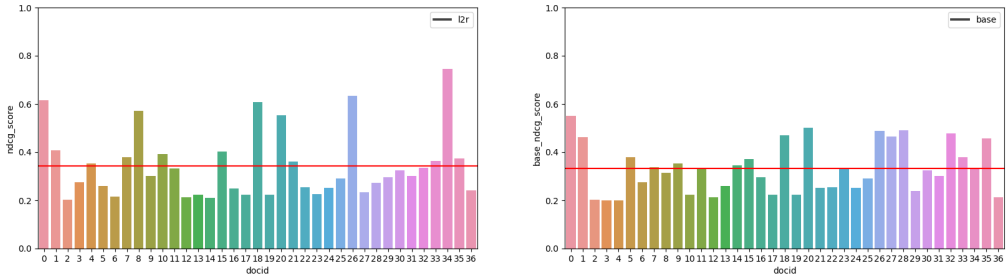

The picture shows all the categories with non-zero importance. We can find that `short descriptions`, `VIAF identifiers`, and `video clips` serve as good features, but they are overwhelmed by the most non-categorical variables except `tf-title`. Because documents with short description, VIAF identifier, and video clip tend to be more concise, standardized, and informative.

## Q7

- map score




- ndcg score




- **Observations**
  - The left figures are l2rranker, while the right ones are BM25.
  - l2rranker:
    - map: 0.07022555505643227
    - ndcg: 0.3437513365850394
  - BM25:
    - map: 0.04391315973201281
    - ndcg: 0.33207116801551484
  - We can find that l2rranker outperforms BM25.
  - l2rranker does improve BM25 performance in most queries, but in some query like `query_1` and `query_32` in MAP, l2rranker does worse than BM25.

In [ ]:
```
from indexing import Indexer, IndexType
from document_preprocessor import RegexTokenizer
from ranker import *
from tqdm.auto import tqdm
import json
from collections import Counter, defaultdict
from network_features import NetworkFeatures
from l2r import L2RFeatureExtractor, L2RRanker
import gzip
```

```
/Users/haoyang/miniconda3/envs/si650/lib/python3.11/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [ ]:
```
# NOTE: If you want to reload a class/method without restarting the whole notebook, use/modify this
#       code to get the latest changes from your file.  You should change it to whatever file/class
#       you want reloaded. The example is for the l2r file but you can use whatever you want.
from importlib import reload
import l2r
reload(l2r)
from l2r import L2RRanker, L2RFeatureExtractor, LambdaMART
```

## Loading part

In [ ]:
```
# Load in the stopwords

stopwords = set()
with open('stopwords.txt', 'r') as f:
    stopwords = set(map(lambda x: x.strip(), f.readlines()))
print('Loaded %d stopwords' % len(stopwords))
```

```
Loaded 543 stopwords
```

In [ ]:
```
# from indexing import read_dataset
# docid_to_category = {}
# dataset_path = 'wikipedia_200k_dataset.jsonl.gz'
# for doc in tqdm(read_dataset(dataset_path), total=200000):
#     docid_to_category[doc['docid']] = set(doc['categories'])
```

In [ ]:
```
# # Get or pre-compute the list of categories at least the minimum number of times (specified in the homework)
# category_counts = Counter()
# for docid, categories in docid_to_category.items():
#     category_counts.update(categories)
# recognized_categories = set()
# minimum_category_count = 1000
# for category, count in category_counts.items():
#     if count >= minimum_category_count:
#         recognized_categories.add(category)
# print("saw %d categories" % len(recognized_categories))

# # Map each document to the smallest set of categories that occur frequently
# doc_category_info = {}
# for docid, categories in docid_to_category.items():
#     doc_category_info[docid] = list(recognized_categories.intersection(categories))

# with open('doc_category_info.json', 'w') as f:
#     json.dump(doc_category_info, f)
```

In [ ]:
```
doc_category_info = {}
with open('doc_category_info.json', 'r') as f:
    doc_category_info = json.load(f)
    doc_category_info = {int(k): v for k, v in doc_category_info.items()}
```

In [ ]:
```
recognized_categories = set()
with open('recognized_categories.txt', 'r') as f:
    recognized_categories = set(map(lambda x: x.strip(), f.readlines()))
```

In [ ]:
```
import pandas as pd
network_features = {}
# Get or load the network statistics for the Wikipedia link network
networks_stats = pd.read_csv('network_stats.csv', index_col=0)
networks_stats.head()
```

Out[ ]:

| | docid | pagerank | authority_score | hub_score |
|---|---|---|---|---|
| 0 | 12 | 0.000030 | 0.004949 | 0.002441 |
| 1 | 303 | 0.000049 | 0.008640 | 0.004462 |
| 2 | 305 | 0.000006 | 0.000564 | 0.000916 |
| 3 | 307 | 0.000046 | 0.007065 | 0.003142 |
| 4 | 308 | 0.000054 | 0.004040 | 0.002497 |

Q3

In [ ]:
```
# from indexing import read_dataset

# pagerank_docids = set(networks_stats.sort_values('pagerank', ascending=False).head(10)['docid'].values)
# authority_docids = set(networks_stats.sort_values('authority_score', ascending=False).head(10)['docid'].values)
# hub_docids = set(networks_stats.sort_values('hub_score', ascending=False).head(10)['docid'].values)
# pagerank_infos = {}
# authority_infos = {}
# hub_infos = {}

# dataset_path = 'wikipedia_200k_dataset.jsonl.gz'
# for doc in tqdm(read_dataset(dataset_path), total=200000):
#     docid = doc['docid']
#     if docid in pagerank_docids:
#         mark = networks_stats['docid'] == docid
#         doc['pagerank'] = networks_stats[mark]['pagerank'].values[0]
#         pagerank_infos[docid] = doc
#     if docid in authority_docids:
#         mark = networks_stats['docid'] == docid
#         doc['authority_score'] = networks_stats[mark]['authority_score'].values[0]
#         authority_infos[docid] = doc
#     if docid in hub_docids:
#         mark = networks_stats['docid'] == docid
#         doc['hub_score'] = networks_stats[mark]['hub_score'].values[0]
#         hub_infos[docid] = doc
```

In [ ]:
```
# pd.DataFrame(
#     {
#         'pagerank': list(pagerank_docids),
#         'authority_score': list(authority_docids),
#         'hub_score': list(hub_docids)
```

```
#     }
# )
```

In [ ]:
```
# pd.DataFrame.from_dict(pagerank_infos, orient='index').sort_values('pagerank', ascending=False)[['docid', 'title', 'pagerank']].reset_index(drop=True)
```

In [ ]:
```
# pd.DataFrame.from_dict(authority_infos, orient='index').sort_values('authority_score', ascending=False)[['docid', 'title', 'authority_score']].reset_index(drop=
```

In [ ]:
```
# pd.DataFrame.from_dict(hub_infos, orient='index').sort_values('hub_score', ascending=False)[['docid', 'title', 'hub_score']].reset_index(drop=True)
```

In [ ]:
```
for row in networks_stats.iterrows():
    network_features[row[1]['docid']] = row[1][1:].to_dict()
```

Q4

In [ ]:
```
# from indexing import read_dataset
# q4_doc_infos = {}
# dataset_path = 'wikipedia_200k_dataset.jsonl.gz'
# for doc in tqdm(read_dataset(dataset_path, max_docs=10000), total=10000):
#     docid = doc['docid']
#     doc_length = len(doc['text'].split())
#     q4_doc_infos[docid] = {
#         'pagerank': network_features[docid]['pagerank'],
#         'doc_length': doc_length,
#     }
```

In [ ]:
```
# import seaborn as sns
# import matplotlib.pyplot as plt

# # Plot article length versus PageRank using the values for the first 10K documents
# plt.loglog()
# sns.scatterplot(
#     x='doc_length',
#     y='pagerank',
#     data=pd.DataFrame.from_dict(q4_doc_infos, orient='index'),
#     alpha=0.1
# )
```

In [ ]:
```
# Load or build Inverted Indices for the documents' main text and titles
#
# Estiamted times:
#    Document text token counting: 3m40s
#    Document text indexing: 4m45s
#    Title text indexing: 22s
from time import time

preprocessor = RegexTokenizer('\w+')
print('Loading document text index...')
t0 = time()
document_indexer = Indexer.load_index('BasicInvertedIndex_doc')
t1 = time()
print('Done in %.2f seconds' % (t1- t0))
title_indexer = Indexer.load_index('BasicInvertedIndex_title')
t2 = time()
print('Done in %.2f seconds' % (t2- t1))
```

```
Loading document text index...
load index: 100%|██████████| 122784/122784 [00:33<00:00, 3665.71it/s]
Done in 278.47 seconds
load index: 100%|██████████| 90877/90877 [00:00<00:00, 479605.21it/s]
Done in 0.45 seconds
```

In [ ]:
```
import gc
```

In [ ]:
```
# Create the feature extractor and an initial ranker for determining what to re-rank
# Use these with a L2RRanker and then train that L2RRanker model
#
# Estimated time (using 4 cores via n_jobs): 2m40s

feature_extractor = L2RFeatureExtractor(document_indexer, title_indexer, doc_category_info, preprocessor, stopwords, recognized_categories, network_features)
scorer = BM25(document_indexer)
ranker = L2RRanker(document_indexer, title_indexer, preprocessor, stopwords, scorer, feature_extractor)

gc.collect()

# Train the ranker
print('Training ranker...')
t0 = time()
training_data_filename = 'hw2_relevance.train.csv'
eval_data_filename = 'hw2_relevance.dev.csv'
ranker.train(training_data_filename)
t1 = time()
print('Done in %.2f seconds' % (t1- t0))
```

```
Training ranker...
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.003572 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2640
[LightGBM] [Info] Number of data points in the train set: 9604, number of used features: 123
Done in 256.44 seconds
```

In [ ]:
```
evaluation_data_filename = 'hw2_relevance.dev.csv'
```

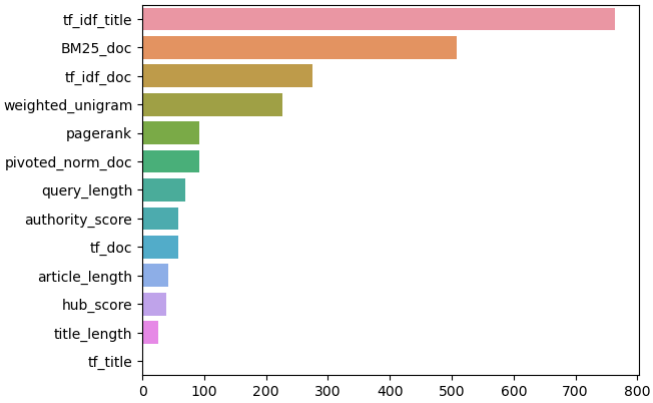
In [ ]:
```
feature_names = [
    "article_length",
    "title_length",
    "query_length",
    "tf_doc",
    "tf_idf_doc",
    "tf_title",
    "tf_idf_title",
    "BM25_doc",
    "pivoted_norm_doc",
    "pagerank",
    "hub_score",
    "authority_score",
    "weighted_unigram",
]

categories = []
for key, val in sorted(ranker.feature_extractor.category_to_id.items(), key=lambda x: x[1]):
    categories.append(key)

feature_names.extend(categories)
```

In [ ]:
```
feature_importances = ranker.model.model.feature_importances_
```

Q5

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
num_cols = 13
# Plot the feature importances with sorted features
x = ranker.model.model.feature_importances_[:num_cols]
y = feature_names[:num_cols]

x, y = zip(*sorted(zip(x, y), reverse=True))
sns.barplot(x=list(x), y=list(y))
```

Out[ ]: <Axes: >




In [ ]:
```
non_category_feature_df = pd.DataFrame(
    {
        'feature': y,
        'importance': x
    }
)
non_category_feature_df
```

Out[ ]:

| | feature | importance |
|---|---|---|
| 0 | tf_idf_title | 764.455770 |
| 1 | BM25_doc | 507.411774 |
| 2 | tf_idf_doc | 274.577948 |
| 3 | weighted_unigram | 226.078532 |
| 4 | pagerank | 92.240010 |
| 5 | pivoted_norm_doc | 92.201091 |
| 6 | query_length | 70.105271 |
| 7 | authority_score | 57.673680 |
| 8 | tf_doc | 57.404420 |
| 9 | article_length | 42.369930 |
| 10 | hub_score | 38.622510 |
| 11 | title_length | 25.506510 |
| 12 | tf_title | 0.000000 |

Q6

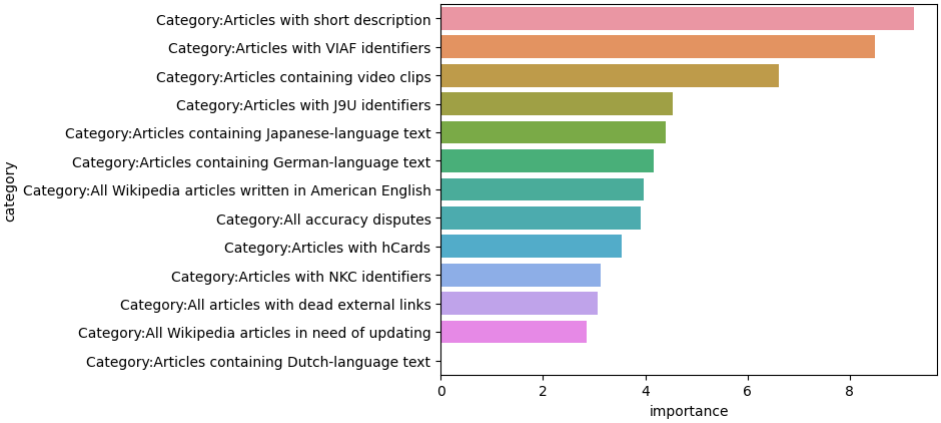
In [ ]:
```
category_length = len(recognized_categories)
category_importance = pd.DataFrame(
    {
        'category': feature_names[-category_length:],
        'importance': ranker.model.model.feature_importances_[num_cols:]
    }
)
category_importance.sort_values('importance', ascending=False).head(5)
```

Out[ ]:

| | category | importance |
|---|---|---|
| 47 | Category:Articles with short description | 9.24552 |
| 79 | Category:Articles with VIAF identifiers | 8.49493 |
| 53 | Category:Articles containing video clips | 6.61918 |
| 92 | Category:Articles with J9U identifiers | 4.53467 |
| 80 | Category:Articles containing Japanese-language... | 4.39246 |

In [ ]:
```
filter_out_zero = sum(category_importance['importance'] > 0)
sns.barplot(x='importance', y='category', data=category_importance.sort_values('importance', ascending=False).head(filter_out_zero + 1))
```

Out[ ]: <Axes: xlabel='importance', ylabel='category'>




Save and Reload the model

In [ ]:
```
ranker.model.save('l2r.model.txt')
```

In [ ]:
```
# ranker.model.load('l2r_a.model.txt')
```

load the dataframe

In [ ]:
```
import relevance
reload(relevance)
from relevance import run_relevance_tests
```

In [ ]:
```
test_filename = 'hw2_relevance.test.csv'

ranker_info = run_relevance_tests(test_filename, ranker)
```

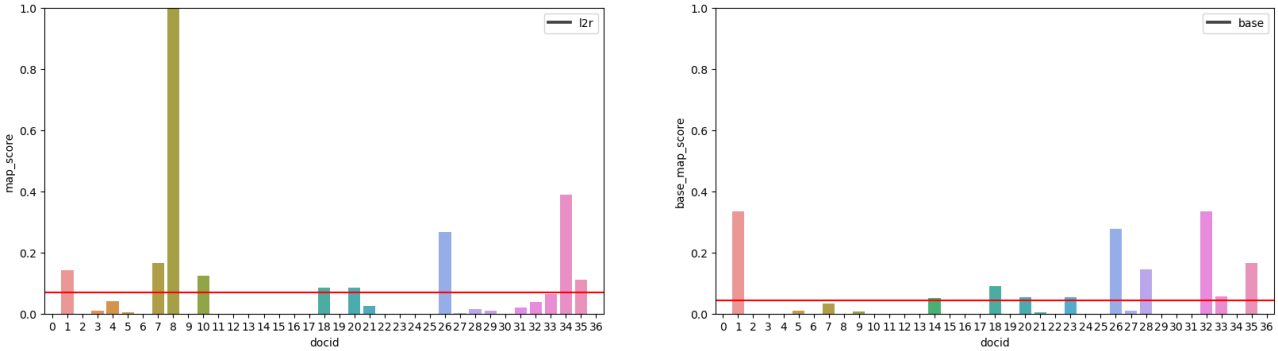
```
  0%|          | 0/37 [00:00<?, ?it/s]100%|██████████| 37/37 [03:10<00:00,  5.14s/it]
```

In [ ]:
```
base_ranker = Ranker(document_indexer, preprocessor, stopwords, scorer)
base_ranker_info = run_relevance_tests(test_filename, base_ranker)
```

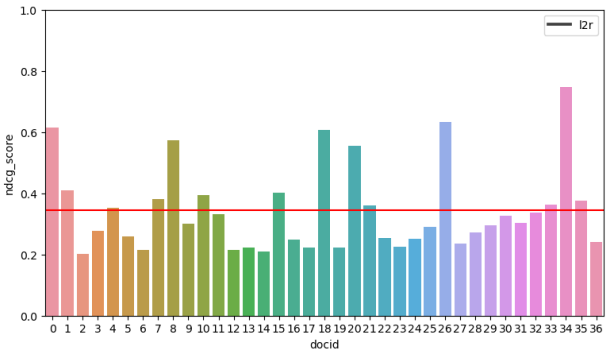
```
  0%|          | 0/37 [00:00<?, ?it/s]100%|██████████| 37/37 [01:51<00:00,  3.02s/it]
```

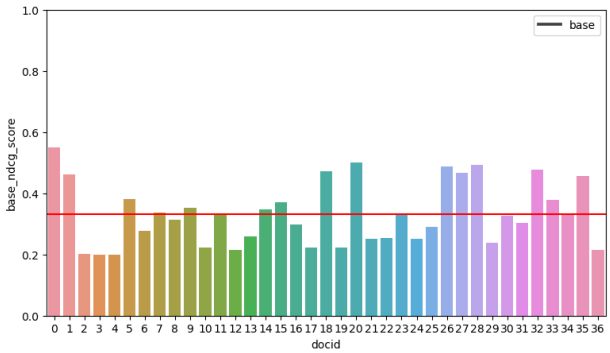
In [ ]:
```
score_df = pd.DataFrame(
    {
        'docid': range(len(ranker_info['map_scores'])),
        'map_score': ranker_info['map_scores'],
        'base_map_score': base_ranker_info['map_scores'],
        'ndcg_score': ranker_info['ndcg_scores'],
        'base_ndcg_score': base_ranker_info['ndcg_scores'],
    }
)
```

In [ ]:
```
# plot the MAP scores for the base ranker and the L2R ranker with a mean line and confidence intervals
# make the same yscale for both plots
plt.subplots(1, 2, figsize=(20, 5))

plt.subplot(1, 2, 1)
sns.barplot(x='docid', y='map_score', data=score_df)
plt.axhline(score_df['map_score'].mean(), color='red')
plt.ylim(0, 1)
plt.legend(['l2r'])
plt.subplot(1, 2, 2)
sns.barplot(x='docid', y='base_map_score', data=score_df)
plt.axhline(score_df['base_map_score'].mean(), color='red')
plt.ylim(0, 1)
plt.legend(['base'])
plt.savefig('map_scores.png')
```




In [ ]:
```
plt.subplots(1, 2, figsize=(20, 5))

plt.subplot(1, 2, 1)
sns.barplot(x='docid', y='ndcg_score', data=score_df)
plt.axhline(score_df['ndcg_score'].mean(), color='red')
plt.ylim(0, 1)
plt.legend(['l2r'])
plt.subplot(1, 2, 2)
sns.barplot(x='docid', y='base_ndcg_score', data=score_df)
plt.axhline(score_df['base_ndcg_score'].mean(), color='red')
plt.ylim(0, 1)
plt.legend(['base'])
plt.savefig('ndcg_scores.png')
```




In [ ]:
```
ranker_info['map'], ranker_info['ndcg']
```

Out[ ]: (0.07022555505643227, 0.3437513365850394)

In [ ]:
```
base_ranker_info['map'], base_ranker_info['ndcg']
```

Out[ ]: (0.04391315973201281, 0.33207116801551484)