An Artwork Search Engine for the Metropolitan Museum of Art
Fall 2022 SI 650 Final Project

Wenjie Wu
University of Michigan - Ann Arbor
wuwenj@umich.edu

Ruqin Chang
University of Michigan - Ann Arbor
ruqin@umich.edu

Abstract

The Metropolitan Museum of Art (MET) is a world-renowned museum located in New York City that features a diverse collection of artworks spanning over 5,000 years. Despite its reputation, the MET's current search engine has limited capabilities and is not user-friendly, making it difficult for visitors to find specific artworks or information about them. To address this issue, our project develops a new search engine interface for the MET that utilizes advanced information retrieval machine learning algorithms and natural language processing techniques, as well as additional features such as cross-language search, query expansion, misspelling correction, and ranking by relevance. We use artworks' metadata and introductory description provided by MET to build the IR system using learning-to-rank techniques and develop a web interface using Flask to allow users to interact with the search engine and receive the top 5 relevant results. Comprehensive model evaluation has shown that our efforts have significantly improved the accuracy and efficiency of the baseline model.

Keywords: Information Retrieval; Natural Language Processing; Machine Learning; Flask

1 Introduction

The Metropolitan Museum of Art (MET) presents over 5,000 years of art worldwide for everyone to experience and enjoy. Since its founding in 1870, The MET has always aspired to be more than a treasury of rare and beautiful objects. Art comes alive daily in the museum's galleries, exhibitions, and events, revealing new ideas and unexpected connections across time and cultures.

We've seen this museum many times in the movies, so we've been yearning for it. We plan to travel to New York City and the MET. But when we use the search engine of the MET website, we believe it could be further improved by using the knowledge we learned in the Information Retrieval class. Currently, the MET's search engine has limited capabilities and is not user-friendly, making it difficult for visitors to find specific artworks or information about them. The search engine of MET only accepts exactly correct keywords to return the name of the artwork. So if we only remember what objects appear in artworks, we cannot get what we want. Sometimes even if we input a valid keyword, the results are still unrelated and irrelevant with random sorting order. Moreover, if we only know the name of artworks in other languages, such as Chinese, Spanish, and Japanese, we will not get any results from the built-in search engine. MET is one of the largest and best museums in the world, attracting many tourists from all over the world every year. At the same time, many researchers from all over the world also use the MET website and related resources of MET as research data sources. Therefore, it is vital to have an accurate search engine that supports multilingual search.

Our project aims to develop a better search engine for artworks in the MET. Our approach is different in that we have focused on improving the MET's own search engine by implementing advanced search algorithms and natural language processing techniques. We have also included additional features such as cross-language search, query expansion, misspelling correction, ranking the results by relevance, and a simple web interface to interact with the search engine. We not only use the name of artworks but also use the attributes such as the object in the artworks, description of the artworks, artists' names, tags, cultures, etc. to build the search engine. We use the BM25 as our baseline and the results of our efforts have shown significant improvements in the accuracy and efficiency of the search engine.

2 Data

2.1 Original Data Source

This project uses the Metropolitan Museum of Art Collection Open Access datasets. It allows us to access the most up-to-date metadata and images of artworks in the Met collection. The Metropolitan Museum of Art provided more than 470,000 artworks in its collection, and the dataset is available for download on GitHub in CSV format. Metadata for artwork includes 54 attributes indicating the artwork's background and history, for example, title, culture, period, reign, and artist. The full list of variables is on the official documentation.

2.2 Web Scraping

Besides the information offered by the dataset, we used the Selenium package to scrape the descriptions of the artwork, if available, from the artwork's page on metmuseum.org to build a text-based information retrieval system. Considering the long execution time to scrape the entire dataset and limited computational resources, we only focus on four object types: photograph, drawing, painting, and figure. Figure 1 shows the number of instances of each type.

Object Type	# of artworks	# of artworks with description
Photograph	28458	10916
Drawing	25788	6704
Painting	5932	3195
Figure	3030	856
Total	**63208**	**21671**

Figure 1: Table of Web Scraping Statistics

2.3 Query Annotation

Unfortunately, the data doesn't have a relevant score for query-document pairs. The data annotation process helps us identify critical additional features and specific Information Retrieval tasks for improvement that our system needs to solve beyond a simple BM25 system. We performed data annotation on 55 queries on the whole dataset and provided 5-point scaling relevance scores for the top 100 BM25-retrieved documents for the query.

The diverse set of queries tests the proposed and desired abilities on a baseline model. These queries can be identified in different groups: spell check, query expansion, cross-language, time, region, culture, department, artist name, artist gender, artwork classification, the object in art, painting type, and complex search. The search queries are included in Appendix A.

The baseline model only uses the artwork's description scraped from the metmuseum.org pages without using additional metadata from Open Access datasets. Following are some findings for model improvement:

- The baseline model doesn't perform well on misspelled words. Spelling correction is a must-have for our artwork search engine.
- Search of the artist's name returns many irrelevant artworks, and the specific artist's works rank poorly in the top retrieved documents.
- Cross-language search is not available in the baseline model.
- Search of a specific year returns a limited number of matched artwork created in that year. Searching for a phrase that describes a specific century returns mostly relevant documents.
- When searching for a specific object in the artwork, the model must rely on description to include the specific word.
- The IR system performs well on single-word queries. However, when dealing with bi-word phrases, the IR system will return irrelevant results to the end user's goal since it cannot weigh the importance of specific word terms. For example, the model returns irrelevant documents with the word "culture" when searching for "Japan Culture".
- The system doesn't do well on complex queries, for example, a question like "What is Claude Monet's most important contribution to art".
- Since the current IR model is built based on the Description of the artwork, if we search for other features in the dataset, the search engine will not meet our expectations. For example. If we search for "Sculpture", the system will return paintings with "sculpture" in the description.

From data annotation distribution, the baseline model gives most results a score of 3 out of 5, which means most keywords only appear in the description of the artworks. There are also some

missing results. It is possible the system cannot retrieve any information. The second largest score we gave is 2 out of 5, which means the keywords only partially matched the description. For example, if we search "European Paintings", some results only contain "European" or only contain "Paintings". The system will return paintings from other countries or other types of art from European. There are still some irrelevant result returns, which we expected should not exist. Surprisingly, the search engine performs well in searching for the object in the artwork. For example, if we search "bird", some results do have a bird in the artwork. But we believe that it is because the keyword is in the description. We assume that there might have more artworks with a higher relevance score that can be retrieved. The distribution of data annotation is shown in Figure 2.

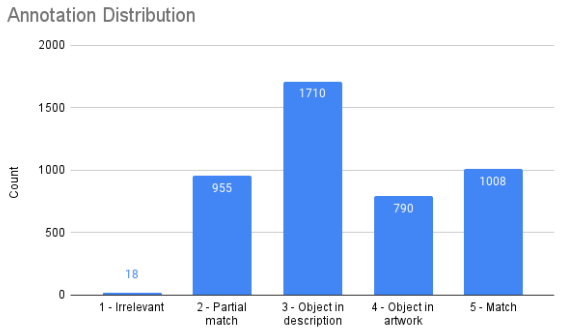

Figure 2: Annotation Distribution

3 Related Work

There is no previous work directly related to ours, but papers with similar methodology. exist One previous paper highlights the algorithm of the XBOX recommender system (3). They build a bilinear model to find the pattern and similarity. Their evaluation method is different from traditional methods like root means squared error. Instead, each test user keeps one item in the test set and trains on the rest of the items. After training, they randomly pick another item for every user, which is not present in that user's training or test items. And then evaluate the model by using the precision score. This work provides us a general insight into how to do a recommendation system, and what is a good recommendation system.

The second paper introduced a neural network architecture for a visually-aware recommendation of art images (2). They perform the optimization to learn the parameters which reduce the loss function by stochastic gradient descent using the implementation in Tensorflow. They evaluate the performance using AUC, normalized discounted cumulative gain (NDCG@k), Precision@k, and recall@k. They choose both a small k, top 20 results, and a large k, top 100 results. They think top-k ranking metrics measured at higher values of k are especially robust to biases such as sparsity and popularity biases. After reading this paper, we decide to use the same evaluation as theirs.

The third paper discusses a semantic search engine for tagged artworks based on word embeddings (6). They use the two-word embedding techniques word2vec and doc2vec to train and evaluate to create a semantic artwork search engine for the dataset. Their search engine can automatically detect and correct spelling errors and typos by using several NLP techniques. The IR model they used is the vector space model (VSM). This paper gives us a direction on how to do misspelling corrections.

The fourth paper concentrates on cross-language search (1). They add a query translation module to generate translated and disambiguated queries and use the translated query as input into a search engine. The user's native language is first translated into the target language, using a bilingual dictionary. The obtained translation candidates are disambiguated, using term cooccurrence statistics, and then passed to the search engine. They used many ways to measure the co-occurrence, such as Mutual Information, Modified Dice Coefficient, Log Likelihood Ratio, and Chi-Square Test, to query term disambiguation. This paper shows us how to build a cross-language model.

The last paper is an improvement of the BM25 model (5). They introduce several variants of BM25 to fit different situations. Furthermore, they also introduce other language models, such as Pitman-Yor process smoothing. For each model, they list the formula and limitations. At the end of the paper, they compare the results of the different models they mentioned. This paper gives us a general idea about the BM25 and its variants, providing the variants we can try in our model.

4 Methods

4.1 Libraries

In this project, we use several libraries in Python to implement our proposed feature of the search engine, including web scraping, data manipulation

and analysis, machine learning, and natural language processing. The used libraries are listed below:

- Selenium: a library that allows developers to automate web browser interactions, which can be useful for tasks such as web scraping or testing web applications.
- BeautifulSoup: a library for parsing and navigating HTML and XML documents, which can be helpful for extracting data from web pages.
- Pyterrier: a library for search engine development, which is used for implementing advanced search functionality and machine-learned ranking techniques.
- NumPy and Pandas: libraries for scientific computing and data analysis, respectively, which can be used for tasks such as manipulating and analyzing large datasets.
- FastRank: a library for training and evaluating ranking models for information retrieval. It provides a fast and flexible interface for working with ranking models and can be used to build models for tasks such as web search and document retrieval.
- NLTK (Natural Language Toolkit): a library for working with human language data (text). It provides tools for tasks such as tokenization, stemming, and part-of-speech tagging, as well as for working with corpora and text classification.
- Googletrans: a library that allows you to translate text using the Google Translate API. It provides a simple interface for translating text from one language to another.
- Jamspell: a library for correcting spelling mistakes in the text. It uses a combination of machine learning and dictionary-based techniques to identify and correct spelling errors.
- Flask: a microweb framework for Python. It allows you to build web applications quickly and easily, using Python as the backend language. It is lightweight and easy to use, making it a popular choice for building web applications and APIs.
- XGBoost, RandomForestRegressor, LightGBM: machine learning techniques to build a more powerful search algorithm that can better understand the content of the artworks and match them to user queries.

4.2 Baseline Model

With only artwork intro description information, we consider 4 ranking models when choosing a baseline model: term frequency (TF), term frequency–inverse document frequency (TF-IDF), BM25, and pivoted length normalization (PL2). BM25 has the best performance and is chosen as the benchmark and model for data annotation. In the following section, we will discuss the single-feature search model and its limitations. Taking advantage of additional metadata available in our dataset, we also incorporate learning-to-rank algorithms and build multivariable models. BM25 model is set as the baseline in both cases.

4.3 Single Feature Search Model

We build a basic search engine first only using descriptions of the artwork we scraped from other websites. We give the model a query and the model should return a list of relevant artworks' document IDs in a descending order based on the relevance score. Figure 3 is the visualized process of our search engine. In this model, we first create the index of artwork description and then implement features in the query reformulation module.

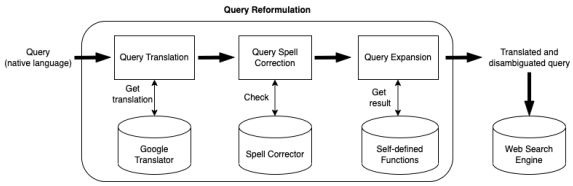

Figure 3: Query Reformulation Flow Chart

4.3.1 Cross-Language Search

Our search engine can support search by any mainstream language in the world. First, we tokenize the query that the user typed in, remove all stop words, and convert the query into lower cases. When the user inputs a query in any native language, the system will translate the query through Google Translate and return a translated English query. However, due to the limitation of the translator and our designed progress, we cannot perform spell correction and query expansion simultaneously. In other

words, if the user types wrong words in other languages, our model cannot do spell correction and query expansion for the user.

4.3.2 Spell Correction

After we get the query in English, we need to ensure the spell is disambiguated. Without spell correction, the search engine will not return any results. For example, if we search "Picaso" instead of "Picasso", the system will get confused and throw a 0 match. To implement this function, we combine natural language processing techniques and a well-developed spell-check library to check the spelling of the user's input query. Instead of a naive approach of correcting each token within a query, the JamSpell library considers words' surroundings and contextual information for better correction. After applying the spell corrector, the system will replace misspelled words with correct words and return a disambiguated query.

4.3.3 Query Expansion

The query expansion feature helps users to browse different aspects of search queries that may be useful for cases where the user is searching for an unfamiliar topic. To implement the query expansion function, we build a pipeline to perform the self-defined functions (4). If the user inputs only one word to search, our system will automatically apply query expansion for the user. There are a few steps in the pre-processing stage, such as tokenizing, lemmatization, and stemming, to get the synsets. The second step is part-of-speech (POS) tagging. Since there are different synsets for the same word depending upon the POS tag, it is necessary to do POS tagging. The whole process of query expansion is shown in Figure 4. Although query expansion can improve the user experience significantly, when we add this function to our search engine, the performance becomes extremely low. So we give up on implementing this feature for now.

4.3.4 Learn To Rank

After the query is ready, we start to optimize the ranking of search results based on the relevance of the documents to the query. We develop seven different algorithms to improve the accuracy of search results: Term Frequency, BM25, TF-IDF, Random Forest, Fast Rank, LightGBM LambdaMART, and XGBoost LambdaMART. PyTerrier library supports complex learning to rank pipeline formulation. Extra features are computed on candidate documents using specialized transformers. The custom transformer also allows us to rewrite queries and achieve misspelling correction, query expansion, and language translation. Random Forest, Fast Rank, and LightGBM are integrated with conventional retrieval transformers supported by PyTerrier. Compared to the baseline model, although our system only performs a little bit better than the baseline on the evaluation indicators, the new system does support many features that were not supported by the old system.

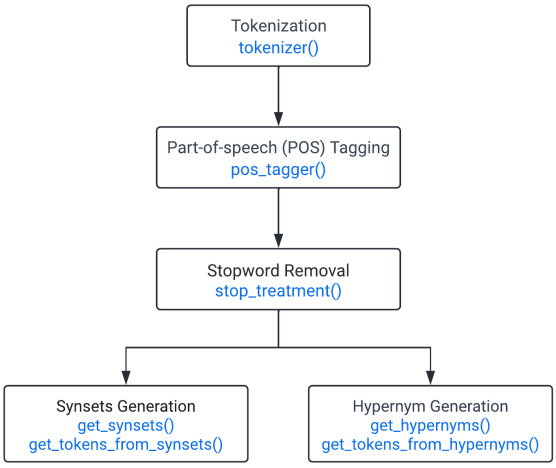

Figure 4: Query Expansion Function Structure

4.4 Multi-feature Search Model

The performance of the single-feature search model is already pretty good. However, if we search for the object or other attribute in the artwork when the attribute does not appear in the description, the search engine will not match the document as we expected. So we introduce the improved model.

To support the user to search for more information related to the artwork, we choose several more features to add from the attributes provided in the MET API. After we tried and evaluated several combinations and took careful consideration, we finally decide to use the combination of 'artwork_intro_desc', 'Department', 'AccessionYear', 'Object Name', 'Title', 'Culture', 'Period', 'Portfolio', 'Artist Display Name', 'Artist Nationality', 'Object Date', 'Medium', 'City', 'Country', 'Region', 'River', 'Classification', and 'Tags'. Those attributes contain all necessary background information, such as cultural background, country information, geographical information, and the artist's personal background. So in our improved model,

we add selected features to the existing model and go through all same steps and use the best algorithm as our final model.

4.5 User Interface

To allow users to better understand and interact with our search engine, we have created a website as our user interface using Flask, HTML, and Bootstrap. Users can enter a search query in any language and receive a list of the five most relevant artworks with information about the artwork title, artist, date, medium, gallery number, and description. Users can also easily access links to artwork pages at metmuseum.org. We record a demonstration of the interactive search engine.

We use the Random Forest in the multi-feature search engine as our core algorithm for the website. The system is able to handle cross-language translation and spell correction. Unfortunately, we are unable to demonstrate the query expansion function supported by the model since we have difficulty importing the Jamspell library on our local Macs. The architecture of our web interface is shown in Figure 5.

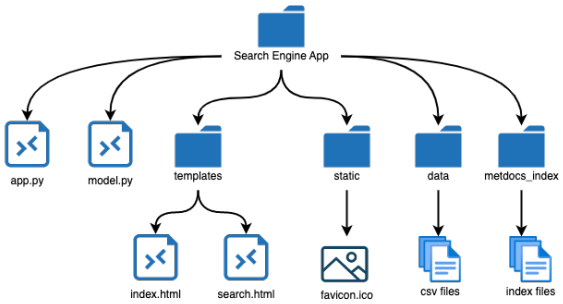

Figure 5: Flask Structure

5 Evaluation and Results

5.1 Evaluation Methods

In this project, we perform 6 different evaluation methods: Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (NDCG) and calculated on the top 10 ranked items (NDCG_cut_10), Precision on the top 10 ranked items (P_10), Recall on the top 10 ranked items (recall_10) and Mean Response Time (MRT). In general, NDCG is a good metric to use when the goal of the ranking model is to return a list of highly relevant items to the user, while MAP is a good metric to use when the goal is to return a list of items that is as precise as possible. Since our goal of the project is to search for the result exactly and sort by relevance, we combine the NDCG and MAP as our primary evaluation methods.

5.2 Model Evaluation

As mentioned in the previous section, we recognize the limitations of the single-feature search engine and build multi-feature models for more comprehensive search needs that an end-user would encounter. Models are evaluated based on the about 4500 labeled query-document pairs created in data annotation. 55 queries used in data annotation are not comprehensive enough to capture the benefit of learning from multi-features. Therefore, the models will be evaluated in two different circumstances and BM25 is the benchmark.

Figure 6 shows the model performance for single-variable models. Since the description of artworks already contains sufficient text, some of them even contain the artist's name and other information, the performance of our baseline models is pretty well. After we added all the functions mentioned in the Methods section and only used artwork descriptions in our models, the evaluation table for the single-variable model is shown in Table. Fastrank model has the highest performance among all, showing an increase of 0.04 in MAP and 0.05 in NDCG.

index	name	map	ndcg	ndcg_cut_10	P_10	recall_10	mrt
0	TF	0.5138	0.6814	0.5029	0.5765	0.1369	10.4766
1	TF-IDF	0.8423	0.8438	0.7916	0.8863	0.1695	9.5194
2	BM25	0.8441	0.8440	0.7914	0.8863	0.1695	8.7452
3	PL2	0.8416	0.8439	0.7929	0.8863	0.1695	8.6930
4	Fastrank	0.8859	0.8989	0.8440	0.9314	0.1740	95.8329
5	Random Forest	0.8562	0.9029	0.8942	0.9353	0.1764	88.8068
6	LambdaMART (xgBoost)	0.8479	0.8777	0.7942	0.9059	0.1712	79.8604
7	LambdaMART (LightGBM)	0.7961	0.8549	0.7122	0.8157	0.1637	86.5183

Figure 6: Evaluation Matrix of Single Feature Model

The performance of our models does improve slightly after using other algorithms. But we still need to need more attributes in our model to provide more insights. After including additional information in documents, the baseline model shows a significant drop in performance. Our models beat the baseline model considerably with a huge improvement. The random forest model specifically has a huge increase in NDCG compared to the baseline model, from 0.8141 to 0.9152, as shown in Figure 7.

Figure 8 and FIgure 9 show the models' performance differences for different types of models in terms of MAP and NDCG. It is important to note that we should compare the model performance

index	name	map	ndcg	ndcg_cut_10	P_10	recall_10	mrt
0	TF	0.5038	0.7232	0.5044	0.5510	0.1251	13.2322
1	TF-IDF	0.6916	0.8306	0.7636	0.8157	0.1491	10.6042
2	BM25	0.6842	0.8141	0.7477	0.8039	0.1479	10.5532
3	PL2	0.6888	0.8275	0.7414	0.7961	0.1472	10.3981
4	Fastrank	0.7271	0.8747	0.8040	0.8529	0.1528	96.9076
5	Random Forest	0.8210	0.9152	0.8687	0.9078	0.1668	105.5147
6	LambdaMART (xgBoost)	0.6666	0.8390	0.6744	0.7529	0.1465	98.9486
7	LambdaMART (LightGBM)	0.6853	0.8574	0.7140	0.7725	0.1455	109.3641

Figure 7: Evaluation Matrix of Multi-feature Model

based on the model types. Overall, Fastrank and Random Forest models perform well using a single feature. Random forest is the best performer among multi-feature models and single feature with query expansion feature. TF-IDF model shows the best performance among multi features with query expansion.

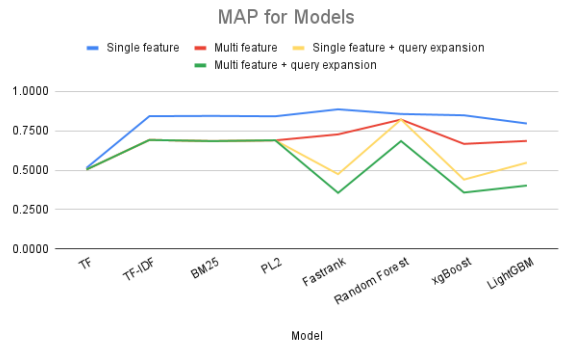

Figure 8: MAP for Models

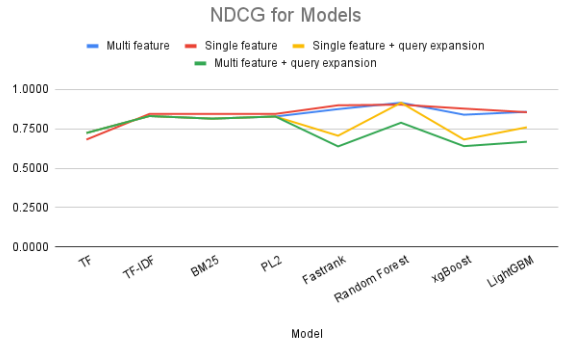

Figure 9: NDCG for Models

5.3 Final Result

From the evaluation above, we will use the Random Forest in the multi-feature search model to build our final model. Although the evaluation of the single-feature search model is higher in all metrics, we want users to get a comprehensive user experience that returns the data they want regardless of the type of attributes they search for. If we only use the artwork description in the model, users are highly likely to get a null return if they use years of artwork, name of artist, cultural background, etc. as the search query.

6 Discussion

Overall, our search engine performs in the way we expected. As art fans, the new search engine can satisfy the essential needs compared to the baseline model. We can use the year of art, the artist's name, the object that appears in the art, the cultural background, and definitely the title of the art to do a search and get what we want most of the time. The difference between the baseline model and our final model proves that those four features do contribute to our system. However, since our potential user also includes researchers and other professionals, there are still some limitations.

The biggest limitation is we have limited data for the current search engine. Although there are around 500 thousand artworks with 54 features in the original dataset, there are only 20 thousand artworks have descriptions. There is also a data integrity issue in the original dataset. Some of the less-known paintings lack descriptions and other necessary attributes. This issue may lead to poor performance and ranking in our search engine.

7 Conclusion

The user experience of the search engine on the Metropolitan Museum of Art's official website is not satisfying and we believe it is necessary to improve the search performance as the MET is one of the four largest museums in the world. After analyzing the user needs and collecting all the data we need using web scraping and the official API of MET, we decide to add four important features to the new search engine: cross-language search, spell correction, query expansion, and learn-to-rank. In order to allow users to understand and interact with our search engine, we also deploy a simple website using Flask and the user will get the top 5 relevant results.

8 Other Things We Tried

We tried to implement the query expansion as we planned, however, it did not work in the way we expected. This function actually works with no error throw and is integrated with other functions well, but this function has a negative effect on the performance of our search engine. After we apply

this function to the search engine, the performance becomes extremely low. The MAP of our baseline model is around 0.80, while the MAP after applying this function decreases to around 0.002, which is not acceptable. We spent 20 hours on this function to try to figure out and fix the potential problem but still cannot improve the performance of the model. We guess the reason that leads to this failure is the ambiguity in the meaning of words. For example, if we search "van" and expect the system can return "van Gogh" or "Vincent van Gogh". But the system returns words like "truck", "camper", "car" and so on. Because the synonym corpus in NLTK is super broad, not just for art-related words, the returned synonym has no relationship with our expected result and will downgrade the overall performance dramatically.

9 Future Work

If we had a chance to start the project over and got enough time, we definitely will try different ways to do the query expansion function, instead of using NLTK. We already have some ideas that may work but will take a lot of time to implement. We can develop this function from four aspects: thesaurus expansion, co-occurrence expansion, relevance feedback, and pseudo-relevance feedback. Thesaurus expansion can be used to identify synonyms or related terms for the words in the user's query. Co-occurrence expansion involves identifying other words that frequently co-occur with the words in the user's query in the documents in the collection. Relevance feedback involves using the feedback provided by the user (e.g., clicking on certain documents or marking them as relevant or not relevant) to determine which terms are most relevant to the user's needs. Pseudo-relevance feedback involves using the initial search results to determine which terms are most relevant to the user's needs. These terms can then be added to the query to expand its scope, but need a large amount of time if it is impossible to use existing libraries.

If we got enough time, we will also implement a new function - user feedback. The performance of search engines will be boosted if the system can improve the algorithm and ranking automatically after getting feedback from users. Users will apply explicit feedback on the relevance of search results, either by clicking on certain documents or by rating them as relevant or not relevant. This feedback can be used to adjust the ranking of documents in the search results and improve the relevance of future searches. We can also do personalize searching based on user feedback. This involves adapting the search results to the preferences and past search behavior of individual users. For example, the system may prioritize documents that the user has previously marked as relevant or that are similar to documents the user has clicked on in the past. To implement user feedback into an IR system, we will need to design a new user interface that allows users to provide feedback and modify their searches, as well as a new system for collecting and processing this feedback. We will also need to determine how to incorporate the feedback into your search algorithm, either by adjusting the ranking of documents or by modifying the search query.

From the result of the performance, we believe that there is still some space to improve the performance. One thing we can do to polish the performance is to do hyperparameter tuning. We use the default parameters when training Random Forest, Fast Rank, LightGBM LambdaMART, and XGBoost LambdaMART. The next step to improve the performance is to find the best parameters for each model. We expect after tuning the hyperparameter, the performance will be boosted.

References

[1] MAEDA, A. Query term disambiguation for web cross-language information retrieval using a search engine: Proceedings of the fifth international workshop on information retrieval with asian languages. https://dl.acm.org/doi/10.1145/355214.355218, Nov 2000.

[2] MESSINA, PABLO, E. A. Curatornet: Visually-aware recommendation of art images. http://ceur-ws.org/Vol-2697/paper2_complexrec.pdf.

[3] NOAM KOENIGSTEIN, N. N. The xbox recommender system. https://www.researchgate.net/publication/254464376_The_Xbox_recommender_system, Sep 2012.

[4] SWAROOP, S. A simple query expansion. https://medium.com/@swaroopshyam0/a-simple-query-expansion-49aef3442416, Jan 2020.

[5] TROTMAN, A. Improvements to bm25 and language models examined: Proceedings of the 2014 australasian document computing symposium. https://dl.acm.org/doi/abs/10.1145/2682862.2682863, Nov 2014.

[6] WEBLER, J. Art2vec: a semantic search engine for tagged artworks based on word embeddings. https://www.en.pms.ifi.lmu.de/publications/projektarbeiten/Jaderson.Webler/PA_Jaderson.Webler.pdf, Feb 2019.

10 Appendices

A Data Annotation Query

qid	type	query
q1	Query expansion	Van Gogh
q2	Artist name	Vincent van Gogh
q3	Spell check	Voncent van Gogh
q4	Query expansion	Van
q5	Spell check	Monet
q7	Artist name	Claude Mnaet
q8	Spell check	Claude Manet
q9	Spell check	Picasso
q10	Spell check	Pablo Picasso
q11	Spell check	Picaso
q12	Cross-language	梵高
q13	Cross-language	鸟
q15	Time	1805
q16	Time	1649
q17	Time	1887
q18	Time	2020
q19	Time	seventeenth century
q20	Time	seventeenth-century
q21	Time	19th century
q22	Object in art	bird
q23	Object in art	Wheat Field
q24	Object in art	grape
q25	Object in art	Park
q26	Object in art	Portrait
q27	Painting type	Oil
q28	Painting type	watercolor
q29	Classification	photo
q30	Classification	Photograph
q31	Region	Dutch
q33	Region	New York
q34	Region	Bohemia
q35	Region	China
q36	Region	Chinese
q37	Region	Spain
q38	Department	Egyptian Art
q39	Department	European Paintings
q40	Department	Greek
q41	Department	Asian Art
q42	Object in art	Female
q43	Object in art	Male
q44	Gender	Female Artist
q45	Culture	Meissen
q46	Culture	Japan Culture
q47	Culture	American Culture
q49	Classification	Sculpture
q50	Classification	Pottery
q51	Complex query	Who was influenced by Claude Monet
q52	Complex query	What is Claude Monet most important contribution to art
q53	Complex query	the death of the great philosopher Socrates
q54	Complex query	father of Impressionism
q55	Region	Mesoamerica