HIV Search Engine Report

Tianyi Wang
wtyjoyce@umich.edu
University of Michigan
Ann Arbor, MI 48109

Zidan Huang
hzidan@umich.edu
University of Michigan
Ann Arbor, MI 48109

Tianye Wang
tianyess@umich.edu
University of Michigan
Ann Arbor, MI 48109

1. Introduction

AIDS has been a persistent and worldwide public health problem where the virus attacks the body's immune system and causes life-long suffering for people. Due to its severity, people are generally afraid of it, not willing to know more about it and have limited knowledge of AIDS. This could be worse in a less developed world. Thus, we would like to design a vertical search engine that revolves around the policies and laws of HIV in different countries. Our purpose is to help the audience know existing policies in their region when they need guidance or want to seek help from local authorities or to help policymakers compare laws among countries. Before we started, we found a policy finder platform, the HIV Policy Lab, that gathers laws and policies in different countries. Yet, it only provided a few policies to choose from and due to its limited number of policies, it hasn't developed a search engine. Therefore, we hoped our system could contribute to creating more instant, diverse, and relevant search results with the data we collected and the model we built. For the vertical search engine, we annotated some data, used learning-to-rank techniques and tried some deep learning models to create cosine similarity as part of our features. With different strategies we tried, fastrank combined with cross-encoder similarity performed the best, and we then would use fastrank as our model to develop an user input with our search engine. Based on our experience, we would recommend testing out different models with different pipelines with multiple tries for other people when they need to create and evaluate their search engines.

2. Data

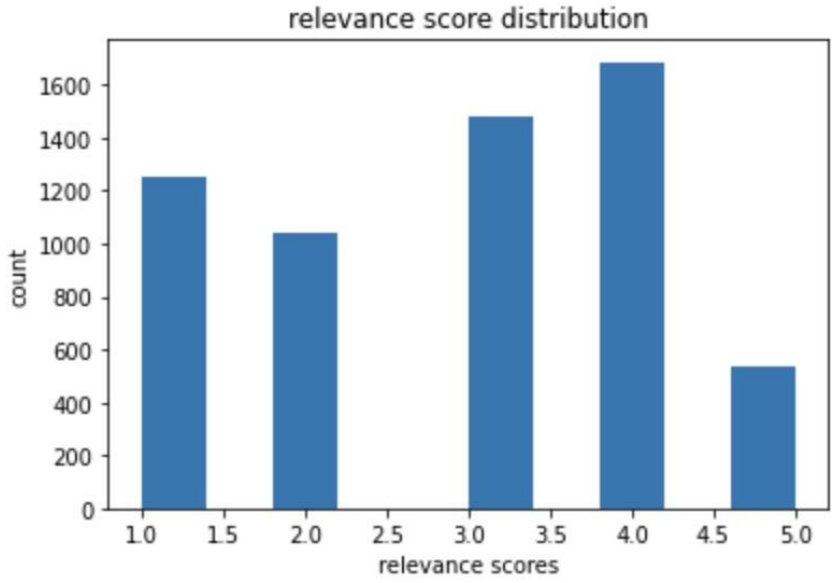
Our data comes from an online platform, Law, and Policies Analytics, which mainly retrieves its data from the National Commitments and Policy Instruments (NCPI) and the WHO Policy Data. There are over 50 topics related to AIDS, such as antiretroviral therapy, condoms, early infant diagnosis, and HIV prevention for different groups of people. Users may select given topics and countries to generate datasets from the website. Since we'd like to study all topics, the scale of the data frame was so large that the website kept crashing. As a result, we decided to focus on policies in less developed regions, the Asian and Pacific areas, with which we were familiar. We downloaded the dataset separately so that we could combine them later.

Firstly, for each dataset, each row represents a country, and each column represents a policy, and the values are specific results of the policy and the country, such as “yes/no” that answer whether the policy exists or some specific time and numbers. There are many empty values in the original dataset, which means no information was collected about a given policy. Thus, we removed all N/As because they reveal nothing. Then, we combined the columns from different datasets into one data frame. Some policy columns each contain a large title and one or two subtitles. We combined the titles and subtitles into a full policy title. For instance, one policy column has the title “Differentiated care services used for pick-up of antiretroviral medicine for community delivery”, and one of its subtitles is “Pharmacy pick-up at the same time as the health facility”. We combined the two into one sentence as our policy. Then, we combined each country and policy, and their corresponding values into rows. This is what our final dataset looks like:

policy
Afghanistan Data quality review conducted to determine accuracy of national level numbers of people on treatment Completed in the last year and results available [UNAIDS National Commitments and Policy Instrument 2022]
Brunei Darussalam Coinfection policies Intensified TB case finding among people living with HIV Yes [UNAIDS National Commitments and Policy Instrument 2019]

Since the original data didn’t have relevance scores for our resulting ‘policy document’, we used TF-IDF, PL2, and TF models to search for 60 queries and select the top 100 documents each and a total of 2000 document results. To generate annotations, which are the relevance scores, we considered how much the document results meet the needs of the queries we created and used a score range from one to five, where one means the least relevant and five means the most relevant. For instance, if the query asks about policy A in country B for a group of people C, then the most relevant results should include policy A, country B, and the group of people C and receive a score of 5. If it only has policy A, then it’s somewhat relevant, and we may assign a score of 2 or 3 to this query.

The graph below shows the relevance score distribution of the annotation. We believe we have a fairly good start as we have the highest distribution of rank 4 which means the search results are partially correct. Yet, we also have a high distribution of rank 1, which we should pay attention to while designing our model.



3. Related Work

In general, we want to construct a vertical search engine on existing policies on AIDS in different countries and times. Yet, we have not found anyone working on the exact problem we do, but as a good start, we found a paper regarding an online platform, the HIV Policy Lab, which serves a similar purpose as our system does. We both understand the importance of comparing HIV-related policies across nations since there are substantial differences in the content of laws and policies among countries (Kavanagh, 2020). Yet, the platform isn't a vertical search engine and it only provides a few topic options to choose from. Thus, we believe by extracting terms, our system may propose more options about policies for the audience. Still, we are building a vertical search engine, so we read a paper that studies the deficiencies of full-text search which helps in recognizing potential problems we may face, such as synonyms and abbreviations (Beall, 2008). Since our topics contain abbreviations such as PMTCT, if the query contains something like "prevention of mother-to-child transmission", then we would not get the right result as expected. Thus, it's something we should consider while building our model. With respect to topic selection which is similar to our policy search in terms of length, we found a paper about question search (Duan, 2008). The authors designed their model about the question topic and focus and employed the framework of language modeling. The difference is that they focused heavily on the structure of the question, but we focused more on extracting words related to the policies and countries we have. Interestingly, they designed a term specificity in describing the information needs of users which is similar to the idea of "tf-idf". For instance, in our case, countries should have a high specificity if the query has specified policy in a certain country. Therefore, we believe this would help improve our model greatly if we have a similar design as term specificity and incorporate it into our model. Since we plan to use learning-to-rank, we read a paper about learning to rank with word features which takes account of the correlation between

words using Supervised Semantic Indexing(Bai, 2010). One of the main differences is that this paper studied features generated by pairing words between the query and the target texts. It is something we would not do considering our limitation of time. Even though they have a mature design of their models, we still found our advantage over the paper which is that we split training and testing data on queries and we have built annotations ourselves. Moreover, we found a paper about privacy policy search engines (Srinath, 2021) which searched for policies based on queries. The topic seems close to ours and the difference is other than just indexing and ranking, the authors spent much time filtering results based on the readability, and vagueness of the policies returned. Fortunately, we don't need to worry much about the readability here as they are not lengthy and vague. Yet, we also believe it would be a good reference if we encounter any policies that are hard to read. Since they mainly used PageRank and query-based document relevance, we believed our approach would produce better results since we would also use learning-to-rank and other deep learning models. To sum up, based on the papers we read, we understood clearly that we should be mindful of abbreviations, and when we design our models, we need to consider the specificity of words. If time permits, we may also learn from Bai's paper although it may take too much work.

4. Methods

Once we have the annotated data, we would like to use learning-to-rank, a key machine learning approach to build models. To work, we need training data to learn our ranking models, and testing data to help us test out model performances. Since our dataset doesn't have the data split, we need to split the new data frame with the scikit-learn library. We couldn't use the get_topics function to directly fetch the right data frame to apply the train_test_split function. Therefore, we took out the 'qid' and 'query' columns from the annotated data as a new dataset which we split into training (60%), valid as (10%) and test dataset (30%).

Then, we trained the models for listwise and pointwise learning to rank. We used a listwise linear technique from FastRank (Foley, 2019), a pointwise regression tree technique, random forests from scikit-learn library, and a listwise regression tree technique LambdaMART from the LightGBM library, and also tried XGBoostRanker from XGBoost library. Meanwhile, we added more features relevant to our dataset hoping that our score could outperform the original baseline from BM25. We considered additional features including the cosine similarity between query and document and the coordinate match score for the query (i.e. how many query terms appeared in the policy). In terms of cosine similarity, we derived it from deep learning models including the bi-encoder model and cross-encoder model. We wanted to use encoders to estimate the relevance of query-document pairs, generate new relevance scores for the pairs, and add the scores as new features. As a result, we derived two sets of cosine similarities from two models and we considered both to be potential features. Then, we constructed several learning-to-rank pipelines with the features listed and passed the pipelines to each of the learning-to-rank techniques to compare the results. Once the models were fit, we could see how well they performed on the test dataset and to compare results with traditional methods like

BM25. Since we have a lot of potential features, we created several pipelines and tested features' performance based on the experiment. For instance, in one pipeline, we included both cosine similarities in the features. We also created two other pipelines where one only included cosine similarity from the bi-encoder model, and the other only included cosine similarity from the cross-encoder model.

Lastly, we created an input box for users to play with, where the audience may be able to input a query, and our system would return the top five relevant policies related. It was designed to mimic a user interacting with a vertical search engine.

5. Evaluation and Results

For the target and evaluation metrics, we chose the two most popular metrics MAP and NDCG, and also NDCG@5 and NDCG@10 which not only considered the difference in relevance scores between items but also how they are placed in the ranking, and"mrt" reflects the response time of different models or how fast they could score the documents.

For our first try, we used features with bm25, TFIDF, and CoordinateMatch for BatchRetrieve. And then, we used train topics to train on fastrank, Random Forest and LambdaMart models and used these models to compare with our baseline models(BM25 and random Performance). By testing the performance on test topics, we noticed that the fastrank has beaten our baseline, but Random Forest and LambdaMart were not performing well in our experiment.

After that, we changed our features and tried to use tf-idf, pl2, and CoordinateMatch as our features. We found the performance of all 3 models went higher, so we decided to use this BatchRetrieve.

	name	map	ndcg	ndcg_cut_10
0	fastrank	0.759344	0.876502	0.707073
1	random forest	0.613418	0.829489	0.625749
2	LambdaMart	0.569892	0.791564	0.533028
3	bm25	0.679230	0.794925	0.647895
4	random	0.520123	0.789584	0.560093

Next, after comparing the cosine similarity from bi-encoder and cross-encoder, we found that the feature from cross-encoder is a better fit for our dataset, so we added cos similarity from bi-encoder as one of our features.

Later on, we tried more models other than these three models. We added the XG Boost to our experiment as well as the BatchRetrieve we got before. We found it performed relatively well. To further improve our model, we tried to use GirdScan and GridSearch to tune the models, but they did not work. Since we encountered difficulties when we used the tuning functions, we

tuned the hyperparameters manually by running experiments and comparing the results. By tuning the models, we got a little bit of improvement on Random Forest and XGbooster. Yet, they were still not able to overperform the fastrank model.

	name	map	ndcg	ndcg_cut_10
0	random forest	0.625784	0.835260	0.635335
1	XGBoost	0.747916	0.849101	0.625500

Thus, we decided to use the fastrank with cosine similarity from the cross-encoder model as well as tf-idf, pl2, and CoordinateMatch as our features in our pipeline to build our search engine. In general, we evaluated the performance of different models by conducting experiments on the testing data. We compared results from different pairs of pipelines and models. The scores for metrics varied slightly, but still, a slight change of feature work in increasing the scores. Here is our best performance result:

	name	map	ndcg	ndcg_cut_10
0	fastrank	0.760175	0.874831	0.702702

As part of our purpose, we would like to build a search engine of related policies to users' search. Thus, we applied the fastrank to our dataset and created an input box for users to search what they are interested in. Our search engine has top 5 related policies as output. Here is an example of our user input and search result:

```
Policy Recommend for Infant Japan

Japan National infant feeding recommendation for HIV exposed infants Yes, replacement feeding
[WHO Policy Data 2021]
Japan Different regimens recommended for high risk infants Yes [WHO Policy Data 2021]
Japan Policy recommendation on point of care early infant diagnosis testing No [WHO Policy Da
ta 2021]
Japan Recommended duration of PMTCT regimen for HIV exposed infants Four weeks [WHO Policy Da
ta 2021]
Japan Recommended regimens for high risk infants AZT, 3TC, NVP [WHO Policy Data 2021]
```

6. Discussion

We believed we did a fairly good job. We tried a lot of new features and parameters. We also tried new methods to find out a most suitable model for our dataset. Generally speaking, NDCG and other evaluation metrics measure the quality and effectiveness of ranking. Since we received a score of 0.87 out of 1 from fastrank, and it outperformed the baseline model by 0.1, we believed our approaches and tries were worth it.

Yet, improvements were still needed for a better search engine model. From the results given, we could see the fastrank model performed the best compared with other models. It also outperformed the performance of BM25 and random performances which are our baselines. But

still, both listwise and pointwise learning to rank models didn't have a significant increase compared with the baseline BM25 model. One reason might be that we use the bm25 and tf-idf functions to fetch documents through our annotation process, and we then use these two functions as our features in the learn-to rank algorithm. The system's results might also vary if different relevant documents were retrieved and then scored. Another reason might be the number of documents is not large enough, which also made it easier for the systems to work on, resulting in the high performance scores for different metrics. What's more, the limited number of annotated data (i.e. ground-truth relevance) would restrict the ability for new models to learn from. Therefore, we believe that we could learn from this experience and have more careful considerations in our future work.

In general, we applied the concept, knowledge, and coding from the Information Retrieval class to our project and used additional approaches outside of class to help build our engine system. We built the search engine system with features like cosine similarity and tried different types of models. Besides the models and functions like tf-idf, pl2, BM25, and CoordinateMatch in BatchRetrieve, as well as learning-to-rank models we learned from class, we also tried some other algorithms like bi-encoder and cross-encoder and used new machine learning models for learning to rank. We would like to have better performance with the help of these new algorithms, but unfortunately, we did not get a better score. We hope we can conclude from what we did and create better search engines in the future.

7. Conclusion

In conclusion, we considered different features such as cosine similarity that we derived from the deep learning models and tried out different learning-to-rank models. After a careful comparison of the evaluation results from different pipelines and models, we found that the learning-to-rank technique, fastrank, performed the best with features including tf-idf, pl2, coordinate match score and cosine similarity from the cross-encoder model. Lastly, we used that model to develop our small search engine, where we give users an input, and then return the top five policies to those users.

Link for Github: https://github.com/wtyjoycee/Hiv_search_engine/

8. Other Things We Tried

Initially, we had many ideas regarding feature selection. For instance, if the query has a certain country like "Japan ", and the returned results have this country, then we wanted to level up the score. It is not the same as tf-idf, because here we only cared about whether the country matched. We were only able to try out whether the country is in a list of countries, but it didn't show that the query returned the exact right country. Moreover, we wanted to know whether the policy exists in a recent year, such as 2022 and 2021. If yes, then we would like to increase the

relevance score. Yet, the apply function in Py-Terrier library didn't work well for us. We spent some time debugging but we were not able to finish it. Some reason might be due to we are not familiar with the doc_scores function and therefore, false to apply that method in our model.

We also tried to use the GridSearch from the scikit-learn library and GridScan function from the py-terrier library to tune the Random Forest and XG Boost, but they did not work. Besides that, we tried machine learning methods like GridSearchCV and KFolds but they still did not work. To save time to do other work, we tuned it manually. Another thing we tried is to use "%100" and "%1000" to get top 100 and 1000 results. It worked on several methods like XGBoost and Random Forest but it significantly decreased the performance score including MAP and NDCG. As a result, we decided not to use "%" in our model.

9. Future Plan

For future work, it would be necessary for us to improve from the beginning, which is to polish our dataset. For instance, we need to solve the problem of abbreviation. For a user searching for both "PMTCT" or "prevention of mother-to-child transmission", he/she should see similar results because they mean the same thing. One possible solution is to include both together in the documents so that the model catches terms from either one. We might also explore an approach such as query expansion that helps recognize more synonyms and related terms to the query. Or, we may explore models like Latent Semantic Indexing that take account of correlations between words such as synonymy (Bai, 2010). Due to the limited time we have, we were unable to explore term specificity, but we believe it might help improve our model later on.

Additionally, to understand more about using the apply function, we found an article where the author wrote a set of functions and placed the function into apply.doc_features( ). We figured that we may write a function as well and place it inside the apply methods, which might help build more features.

Another important issue is we should find out a way to fine tune all our models used rather than to tune it manually. It is important to find out the problem of ERROR provided by GridScan and GridSearch and use them to tune all models.

What's more, we tried to apply the XG Boost model to our learning to rank part, but it did not outperform the fast rank model. Therefore, we decided to try more machine learning and deep learning methods in the future to upgrade our models' performance.

Lastly, what we do in our project is a simple input box for users to input whatever they are interested in and it shows only the top five related policies. In the future, we would like to build a graphical user interface like a web page or a small software for better user experience.

10. Reference

Kavanagh, M. M., Graeden, E., Pillinger, M., Singh, R., Eaneff, S., Bendaud, V., Gustav, R., & Erkkola, T. (2020). Understanding and comparing HIV-related law and policy

environments: cross-national data and accountability for the global AIDS response. *BMJ global health*, *5*(9), e003695. https://doi.org/10.1136/bmjgh-2020-003695

Jeffrey Beall, The Weaknesses of Full-Text Searching, The Journal of Academic Librarianship, Volume 34, Issue 5, 2008, Pages 438-444, ISSN 0099-1333, https://doi.org/10.1016/j.acalib.2008.06.007.

Duan, H., Cao, Y., Lin, C. Y., & Yu, Y. (2008, June). Searching questions by identifying question topic and question focus. In *Proceedings of Acl-08: HLT* (pp. 156-164).

Bai, B., Weston, J., Grangier, D. *et al.* Learning to rank with (a lot of) word features. *Inf Retrieval* 13, 291–314 (2010). https://doi.org/10.1007/s10791-009-9117-9

Srinath, M., Sundareswara, S.N., Giles, C.L., Wilson, S. (2021). PrivaSeer: A Privacy Policy Search Engine. In: Brambilla, M., Chbeir, R., Frasincar, F., Manolescu, I. (eds) Web Engineering. ICWE 2021. Lecture Notes in Computer Science(), vol 12706. Springer, Cham. https://doi.org/10.1007/978-3-030-74296-6_22

John Foley. (2019). FastRank alpha release. https://jjfoley.me/2019/10/11/fastrank-alpha.html