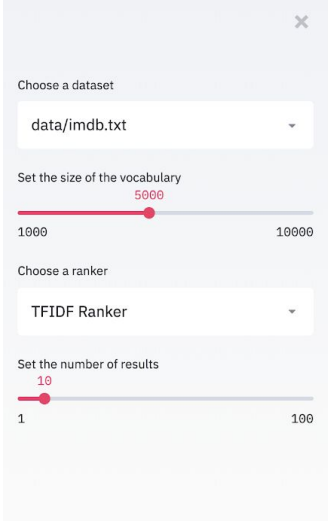


# Quick Search Documentation

## 1 User Guide



The screenshot shows the 'Quick Search' application interface. On the left is a settings panel with a close button (X) in the top right corner. The settings include: 'Choose a dataset' with a dropdown menu showing 'data/imdb.txt'; 'Set the size of the vocabulary' with a slider ranging from 1000 to 10000, currently set at 5000; 'Choose a ranker' with a dropdown menu showing 'TFIDF Ranker'; and 'Set the number of results' with a slider ranging from 1 to 100, currently set at 10. On the right, the search results are displayed under the heading 'Quick Search'. A search bar contains the text 'he needed to save gotham city'. Below the search bar, two results are shown: 'Document: 124 - Score: 0.226' with a description about the Joker and Batman, and 'Document: 253 - Score: 0.198' with a description about Spider-Man and Oscorp.

Choose a dataset

data/imdb.txt

Set the size of the vocabulary

1000 5000 10000

Choose a ranker

TFIDF Ranker

Set the number of results

1 10 100

### Quick Search

he needed to save gotham city

Document: 124 - Score: 0.226

Eight years after the Joker s reign of anarchy, the Dark Knight, with the help of the enigmatic Selina, is forced from his imposed exile to save Gotham City, now on the edge of total annihilation, from the brutal guerrilla terrorist Bane

Document: 253 - Score: 0.198

When New York is put under siege by Oscorp, it is up to Spider-Man to save the city he swore to protect as well as his loved ones

Before running the application make sure that all dependencies from the **requirements.txt** file are installed on your system. The application was developed using Python 3.7.4 but it will probably work with other 3.x Python releases as well.

To run the application type **streamlit run app.py** in your terminal, which will open up a new tab in your browser. The application has a very simple interface. On the left you will find different text retrieval options you can select from.

- **Dataset:** Lists the datasets that are placed in the data folder of the application. For the dataset to be recognized, make sure it's a .txt file that includes every document on its own line. The imdb.txt dataset includes 1000 short descriptions of movies, which is why the file has 1000 rows. Your custom dataset should come in a similar data structure.
- **Vocabulary:** You can set the vocabulary size to use. If your dataset is rather large, it might make sense to set the vocabulary size to a smaller number. The application was not tested with huge datasets larger than 10,000 documents. The default value is 2000.
- **Ranker:** 5 different ranking functions can be chosen from this dropdown. Overall, the TFIDF ranker seems to work best, but especially the semantic ranker might produce some interesting results.
- **Results:** You can also set the number of results you want to see. The default value is 10.

In the main section of the application you will find a text input field where you can type your query. Hitting enter will execute the search and list the top results right beneath it. For each result we show the document ID aka the row in the corpus as well as the score of the respective ranking function.

Switching through different configurations should work rather quickly. However, changing the ranker requires rescanning the dataset, which might take several seconds depending on its size.

## 2 Development Guide

The application was built using the streamlit library. To get an overview of how streamlit works, please take a look at their own [documentation](#). The code of my Quick Search app is divided in two files.

- **app.py** includes most of my code in order to build the GUI application. Streamlit has almost no overhead when it comes to writing template code, which is why the entire application has only roughly 50 lines of code.
- **ranker.py** includes the definitions of the 5 ranking functions that are used within the app. I used two vectorizers from scikit-learn in order to build these rankers. Each ranker comes with an **init** function that will preprocess the dataset according to the respective ranker. The **search** function takes a query and returns the k top results to the user. For future development, one can build on top of this Ranker class.

## 3 Semantic Ranker

4 out of the 5 ranking functions are discussed in class, which is why I'm not describing how they work in detail. However, the semantic ranker was something I came up with, which is why I want to briefly describe how it works.

We start out by calculating the word weights given by the TFIDF ranking function. This essentially tells us how important each word is in the context of the corpus. The TDIDF ranking is also the main reason why removing stop words was simply not necessary for my application. Additionally to the TFIDF weights we also load the GloVe word embedding for each of the words. For more information about GloVe please refer to their [website](#).

king - man + woman ≈ queen



Word embeddings allow us to reason with text numerically. For example, if we take the vector of king, subtract the vector of man and add the vector of woman, we will end up with a new vector that is fairly close to the word queen. To represent a

document using word vectors we could simply sum the vectors of all words in the document. However, this doesn't work very well because non-important words will flood the overall vector with information we don't care about. This is why I used the TFIDF weighting in order to weight important embeddings higher and stop words lower. This gives us a 50 dimensional vector to represent a document. Now, we can compare the query to all the documents in the corpus using the cosine similarity and return the best results.

In practice, I was a little let down by the performance of the ranking function. It doesn't seem to work as well as the TFIDF ranker alone. However, using the semantic ranker we can perform the text retrieval on a semantic level and not on key words alone. For example searching for "labrador" might also return documents that don't include this term but other dog related terms like similar breeds. However, more testing and tweaking needs to be done in order to make this work smoothly.