# CS2108 Project Report

## Problem:

Learning is more and more important today. Students and professionals are expected to learn fast and stay updated. This however, is not easy.

Searching for quality learning material is hard

- Top ranked sites like wikipedia give terse explanations, more suited for experts
- Blogs contain a large noise to content ratio
- Blogs and tutorials sometimes share bad practices, and miss the crux of the material

## Solution:

A search engine for quality learning material.
All material is derived from top ranking university courses released as MOOCs. Most of these MOOCs are filmed live in the university and of the same difficulty as the physical course. i.e , not watered down versions of the course.

Our Data:
MOOC material found on Coursera, by schools such as Princeton and Stanford University. Documents from about 80 courses are used, each with video files, and some with srt (subtitle files), powerpoint, and supplementary material in pdf & text.

Cleaning the data:
Apache tika was used to extract text from srts, pdfs, powerpoint and rtf files, without using optical character recognition (OCR). Other formats were ignored for the following reasons:
Html - usually for files unrelated to subject matter, such as administrative details
Code - does not match well with user queries
Video & images - not useful unless using OCR or sound recognition (difficult to get right)

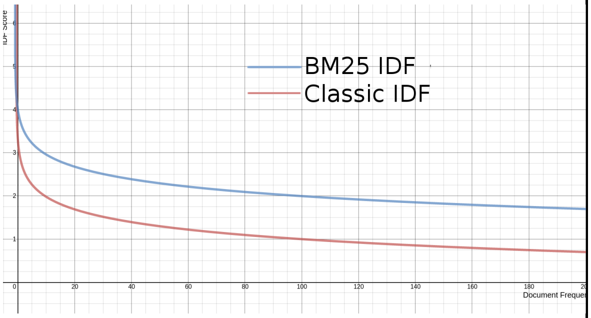
Structure of data stored:

| | |
|---|---|
| Course<br>- Title<br>- Url<br>- Array of documents | Document<br>- Title (ppt, txt, pdf, ...)<br>- Content (extracted content) |

Courses contain many documents. The course title, document title, and document content fields are indexed in our server.

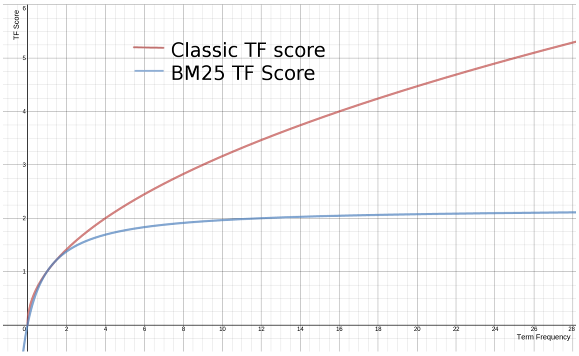
Indexing:
Fields are indexed using using BM25 algorithm. BM25 is very similar to TF-IDF in structure, where score is also like TF x IDF. (using the BM25 versions of TF & IDF)

IDF-score to Document Count for BM25 IDF & Classic IDF



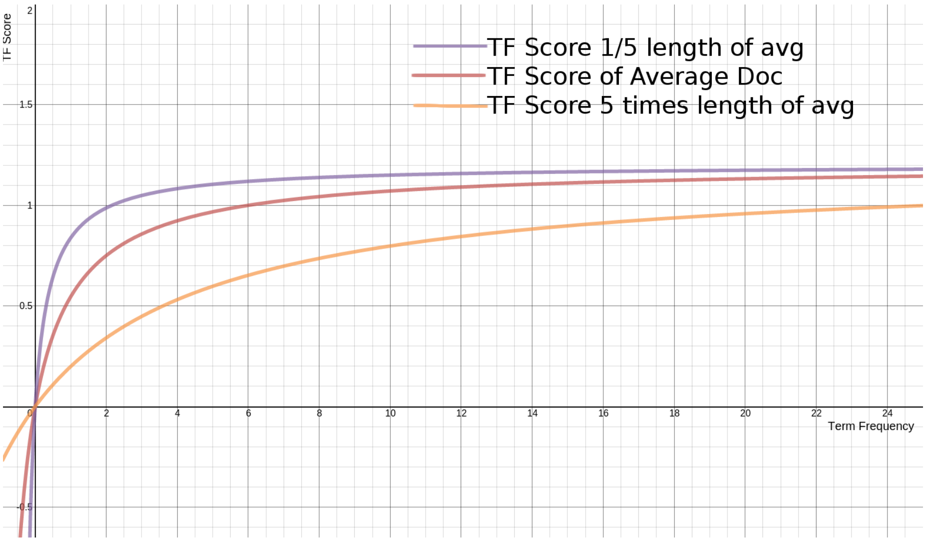
BM25 IDF score & Classic IDF score are very similar

TF-score to Term Frequency for BM25 TF & Classic TF (term frequency value is capped by variables that you can set)



The BM25 TF score reaches an asymptote as term frequency increases above a threshold while classic TF increases without a care. This makes it more accurate than TF-IDF.

TF-score to Term Frequency, for 3 different sized documents



In BM25, document length affects TF. A long document needs higher term frequency to reach the asymptote.

The parameter b (0 <= b <= 1) in BM25 reduces the effect of document length on TF score. A low b makes long & short documents have a similar score. Subtitles can be very short, and lecture notes can be very long but both are of similar importance in this context. As such, we reduced b to 0.25.

Field & Query processing:

We remove stopwords, ignorecase, drop words of size 1, and finally, generate and append length 2 shingles (token sized bigrams) to all indexed fields. The same is applied to queries from the user.

Computing relevance & output format:
Get score for all document content, document titles and course title fields.
For each course, sum up the score of course title and score of matching documents. This score is the final relevance score. The top scoring courses are returned together with excerpts of their matching documents.

Boosting (relative, if all fields are boosted, there is no boost)
The above fields have their scores boosted in the following manner:
Course title: 2 (high boost as the whole course is about the query)
Document title: 1.5 (medium boost as a whole document is about the query)
Document content: 1

## Experiments & Results:

Control: [Filter stopwords, b = 0.25]
Min_length_2: [Filter stopwords, b = 0.25, min_length = 2]
Shingles: [Filter stopwords, b = 0.25, use shingles]
Higher b: [Filter stopwords, b = 0.75]
Stemming (porter2 stemming): [Filter stopwords, b = 0.25, stemming]

Test queries:

1. "Prisoner's dilemma"
2. "What is a neural network"
3. "Time dilation"

The above 3 queries were performed with the 5 settings above. All queries took about 350-500ms to compute. Results were mostly similar and relevant, except for query 1. Only min_length_2 performed with high accuracy. 's' was considered a token in all the other options, resulting in poor matches. Results are biased towards Computer Science topics, from a shortage of other course types.

Our conclusion is that more thorough testing is required, and more edge cases should be included into the test set. The various settings are great only at catching edge cases, so more edge cases are needed to determine their importance. We also need to establish a 'ground truth' to make search relevance quantifiable. It is a tricky and time consuming endeavour since match relevance also varies on how well the user expresses his desired results.

Having sufficient data also plays a big role. 80 courses is insufficient for testing. There may only be 1-2 good matches out of all the courses.

## Other Findings

There are many metrics that contribute to a user's search experience beyond search relevance. Responsiveness too, is one factor. Our web app took a long time to return search results, taking up to 3 or 4 seconds. Our search engine however, reported a processing time of 300-500ms. This is likely due to server latency (server hosted in US), and a low shard count in our ElasticSearch server.

Awareness of all metrics affecting user experience is essential when pushing any app into production.