# Test Lab Document Search Tool

Stephen Mullaly (903548858)
MGT 6748 – Fall 2021

## Problem Statement

Hunter Douglas is the world's leading manufacturer of window coverings as well as a major manufacturer of architectural products. The majority of global testing for window coverings is performed at a facility located in Colorado. This facility handles hundreds of requests for testing each year and produces a large quantity of reports.

Over the last several years a number of systems have been utilized for submitting, assigning, and issuing these reports. This has resulted in no central database of test reports, but reports spread across multiple locations.

The test lab receives regular requests for past reports as research and development work is carried out. These requests can range from very specific requests for a particular test result to very vague in nature to see if anything has been done in the same general area as another request.

The goal of this project was to create a central location for all the past test reports and to create a way for test lab staff to easily locate reports when requests for past results are received. This would be carried out by either conducting a keyword search or by allowing for a test report to be uploaded and conducting a document similarity search based on the upload.

## Data Source

Approximately 7,500 reports covering the last 10 years were collected. Additional reports may be made available in the future and new reports are issued every day. If additional reports are recovered and as more reports are written they will be added to the report search tool.

## Methodology

The first step in this project was to locate all the past report pdf files in one central location. This was done manually as the files were located in two main areas and only required occasional renaming to combine them all into a single folder. Some files that were stored in an SAP database are currently unavailable, but work is being done to release those reports.

Once all the files were in a single folder the first step was to get the reports read into the system and in a usable format. All files in the single folder where the reports were saved were read using the os package in Python. Each file was confirmed to be a .pdf file and then read into the system.

As each file was read in some basic cleaning steps were taken using the Natural Language Tool Kit (nltk) and Regular Expressions (re). The first step was removing line breaks (\n), tabs (\t), and other similar special characters. Then all punctuation was removed, extra white space was removed, and finally all characters were normalized to lowercase. After these initial cleaning steps the document was reduced down to a string containing all of the words in the document. This string was saved as a value in a dictionary with the document id as the key to create the 'corpus', the contents of all documents.

After creating a corpus, additional cleaning tasks were carried out using the nltk package. The first task was to remove stop words that would add very little value to the document as well as removing single letter words. Additional task specific stop words were also added to the standard list of stop words, such as company name, address, email extension, and others. With stop words removed, lemmatization was performed. Lemmatization is used to find the dictionary form of each word in the document, for example, the word 'better' is changed to its lemma form which is 'good'. This step helps to standardize word usage across all documents which were written by many authors. The final step of the cleaning tasks was stemming. Stemming is used to reduce words to the root form. For example, 'walking' and 'walked' would become 'walk'. This step helps to reduce all words to one form so that usage is accurately represented and not diluted across multiple tenses.

After the second round of cleaning tasks, each document is saved as a list with each word an item in the list. All documents are then added to a list of these word lists, which will be used for Natural Language Processing (NLP). The first NLP process is to create a Term Frequency, Inverse Document Frequency (TF-IDF) vector.

The TF-IDF vector combines Term Frequency and Inverse Document Frequency to give each word in each document an importance score. Term Frequency looks at each word in a document and counts the number of occurrences. Since this is dependent on document length, it normalizes these values by dividing the number of occurrences of each word by the total number of words in the document. This value shows what words occur the most in each document. Inverse Document Frequency looks at each word used in the corpus and determines the number of documents that use that word. This frequency is then divided by the total number of documents in the corpus to normalize the values. The inverse is then taken to create the IDF so that a high scoring IDF is a word used in fewer documents, and a low scoring IDF is a common word used across many documents. This acts as a score for how informative a term is. The product of these two values creates the TF-IDF score for a word in that specific document. The TF-IDF vector gives these scores for each word in each document of the corpus. A high TF-IDF score represents a word that occurs frequently in the given document, but infrequently across all documents in the corpus. These high scores give a good indication of what words are important in a document and help determine what documents have similar importance based on how their TF-IDF vectors compare.

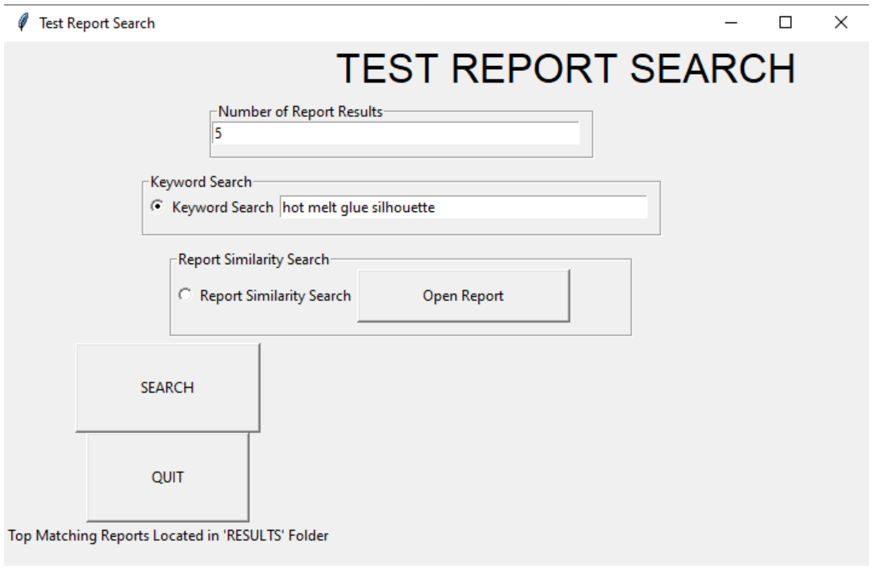
The Scikit-Learn package was used for the Natural Language Processing. When creating the TF-IDF vectorizer in Scikit-Learn it has a number of parameters for the user to adjust. The following parameters were used for this project:

1. lowercase = True: Although the text was normalized as part of data cleaning this was set to confirm that normalization was completed.
2. stop_words = 'english': Although stop words were removed using the nltk package this was used to remove any additional stop words that may not have appeared in nltk.
3. ngram_range=(1,3): This was used so that combinations of words would be recognized if used together frequently. Many products have a two-word name, such as 'Light Lock' or 'Lite Rise' that may be spelled as a single or word or split into two words so this confirms that these are still recognized.
4. max_features=25000: This sets the maximum number of features to use in the TF-IDF vector. This was set at 25000 to balance between inclusion of words with limited use and also speed of search results.
5. min_df = 1: This parameter sets the minimum number of documents a term must be present in for it to be included in the TF-IDF vector. This was set at one because many R&D projects will use a special production number unique to a single test lot so these numbers may be present in a search and should be included in the TF-IDF.
6. max_df = 0.85: This parameter limits features included in the TF-IDF vector to those that appear in fewer than 85% of all documents. This helps to remove words that are common to all documents and do not add value. These can be thought of as project specific stop words.

After the TF-IDF vector has been created the results of these initial cleaning and analysis steps are exported using the pickle library. The exports include a sparse dataframe containing the results of the TF-IDF vector for each document, the Scikit-Learn model of the vectorizer so that documents in the search can be transformed using the same TF-IDF vector parameters, and a list of all document names for recalling files once search results are available.

The portion of code described up to this point can be run independently as new reports are added to the report folder. The exported files will be overwritten so that future searches will cover all current reports without needing to do anything after running the script to clean and process the documents.

The actual search function is carried out using a desktop app made using Tkinter, a GUI package, in Python. The GUI allows a user to specify the number of results to return, select a keyword search and input keywords, and select a similarity search and upload a document to use for similarity calculations. Once the user determines their search criteria they can click on the 'SEARCH' button and results will be returned with the suggested documents transferred to a folder for easy access.



Number of Report Results: 5

Keyword Search: [✓ Keyword Search] hot melt glue silhouette

Report Similarity Search: Report Similarity Search

Top Matching Reports Located in 'RESULTS' Folder

The search is carried out using the same functions that were applied to all the documents that are to be searched. The keywords or uploaded documents go through the previously described cleaning steps and then are vectorized using the TF-IDF vector model that was created using the full library of past reports. Once the specified report or keywords are vectorized then similar documents in the past reports are found using pairwise cosine distance. This allows reports to be compared based on the direction of two TF-IDF vectors. If two vectors have a small distance, this means that their respective documents are made up of similar words with a high TF-IDF score, and this may indicate a similarity in the topics of the two documents.

The resulting document distances are sorted and the closest documents are returned to the user in the number requested. The results are displayed in the GUI as well as copied to a new folder for easy access for the user.

## **Evaluation and Final Results**

When tuning TF-IDF vector parameters, the goal was to have quality results with a search time that was reasonable for the end user. The parameters listed above achieved these goals. Testing was done by taking several random reports and reviewing them to determine keywords that should return the report and confirming that the search successfully did so. With the parameters chosen, the desired reports were returned in search times of approximately 30-60 seconds. For similarity search, several reports were selected for testing and the results were reviewed to confirm that similar reports were returned. Some of the reports selected were random reports, and others represented reports that were part of a series of packaging tests recently performed that should have been returned as ‘similar’. In all cases, the top result was the report entered into the system. This was a positive sign as it should have a perfect similarity score. For the random reports, returned reports all covered similar topics to the input report. For the report from the series of packaging tests, the top returned reports were the other tests in the series. Based on all the test scenarios the reports returned matched expectations and allowed the user to find the report that they were looking for.

The search tool currently runs as a desktop based application and has been shared with a small team of employees in the test lab. After discussions with users and other interested parties, wider use of this tool is desired. In order to accomplish this, the tool will be adjusted to run on the test lab intranet site.

**References:**

1. Christopher D. Manning, Prabhakar Raghavan and Hinrich Schutze, *Introduction to Information Retrieval*, Cambridge University Press. 2008.

2. Ricardo Baeza-Yates and Berthier Ribeiro-neto, *Modern Information Retrieval*, Addison-Wesley Professional, 2011.

3. https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089

4. Mark Lutz, *Programming Python*, O'reilly Media, 2010.