INflow Labs

Extracting Keyword Frequencies from PDF/DOCX Documents

by Shaun Laranjeira, Ryan Malani March 6, 2023 4 min read

Given the size and scale of many of the documents we're meant to read and comprehend, we want to ensure that our time is being allocated to those focused primarily in the same focus and capability areas that we as a company are looking to grow into, which led to this quick side project for us to extract relevant terms and their frequencies in documents using a TF-IDF (Term Frequency-Inverse Document Frequency) algorithm.

Getting Started

For this project, there are a couple of things/packages that you'll need. For reference, we are using M1/M2 Macs at HQ, so we set up our development environment from miniforge[1].

Given that you already have a python environment set up (and if you're using conda like us, you can create a new conda env using the following command:)

```
conda create env --name YOUR_NAME python=3.8 -y
```

We decided to use Python 3.8 as it is known to have more stable builds/alleviate many of the dependency and compatibility of machine learning libraries (SciKit, PyTorch, Tensorflow) on M1/M2 metals. Once you have your new conda environment activate it using:

```
conda activate YOUR_NAME
```

For those using Windows, creating a new virtual environment[2] should act to serve the same purpose. To specify the python version used, you can use the command option --python.

With the environment active, it's time to start installing the requirements that we'll be using for this project with pip. This single line should install all of the desired packages.

```
pip install PyPDF2 docx gensim scikit-learn jupyter
```

Vectorize on Frequency of Words in the Text

First, given that we have a folder of documents in pdf and docx formats, we need to have the ability to parse out the text in these documents and output a vector of the frequency that each word is used in the text. To do so, we're going to use the PyPDF2[3], docx[4], and scikit-learn libraries, with a Term Frequency-Inverse Document Frequency (TF-IDF) Vectorizer algorithm from scikit[5]. The code below allows us to do just that -

```
from sklearn.feature_extraction.text import TfidfVectorizer
import PyPDF2
import docx
import os

documents = []
folder_path = 'documents/'

def extract_text_from_docx(filename):
    doc = docx.Document(file_path)
    full_text = []
```

```
    for para in doc.paragraphs:
        full_text.append(para.text)
    return '\n'.join(full_text)

def extract_text_from_pdf(filename):
    with open(filename, 'rb') as f:
        pdf = PyPDF2.PdfReader(f)
        text = '\n'.join([pdf.pages[i].extract_text() for i in
range(len(pdf.pages))])
        return text

for filename in os.listdir(folder_path):
    if filename.endswith('.pdf'):
        file_path = os.path.join(folder_path, filename)
        text = extract_text_from_pdf(file_path)
        documents.append(text)
    elif filename.endswith('.docx'):
        file_path = os.path.join(folder_path, filename)
        text = extract_text_from_docx(file_path)
        documents.append(text)

# Convert the documents into a matrix representation
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)
tfidf_matrix_array = tfidf_matrix.toarray()

# Access the TF-IDF values
vocabulary = vectorizer.vocabulary_
for i, row in enumerate(tfidf_matrix_array):
    print("Document", i)
    for j, value in enumerate(row):
        if value > 0:
            word = [word for word, index in vocabulary.items() if
index == j][0]
            print("\t", word, ":", value)
```

Extracting the Topic Popularity

In the following code block, we are using preprocessing tools from gensim, to include simple_preprocess and STOPWORDS. Combining this with a list of desired technology words allows us to only extract the relevant terms and their frequencies throughout the document(s). From there, we create a Bag of Words (BoW) corpus which provides a tuple of the ID of the token (word) in the document and it's frequency. Then, use an LdaModel (Latent Dirichlet Allocation Model)[6] to infer the topics contained within the documents.

```
import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS
from gensim import corpora, models
import pandas as pd

# this list is where we copied the capability words from our website
technology_words = []

filenames = [filename for filename in os.listdir(folder_path)]

# Prepare the data
def preprocess(text):
    result = []
    for token in gensim.utils.simple_preprocess(text):
        if token not in gensim.parsing.preprocessing.STOPWORDS and
len(token) > 3 and token in technology_words:
            result.append(token)
    return result

# Convert the documents into a matrix representation
documents = [preprocess(doc) for doc in documents]
dictionary = gensim.corpora.Dictionary(documents)
bow_corpus = [dictionary.doc2bow(doc) for doc in documents]

# Train the LDA model
ldamodel = gensim.models.LdaModel(bow_corpus, num_topics=10,
id2word=dictionary, passes=50)
```

```
# Extract the topics
topics = ldamodel.show_topics(num_topics=10, num_words=10)
# for topic in topics:
#         print("Topic", topic[0], ":", topic[1])

doc_topics_df = pd.DataFrame(columns=['document', 'topic_id',
'summary'])
for i, doc_topics in enumerate(document_topics):
    topic_scores = [score for _, score in doc_topics]
    if topic_scores[0] > 0.4:
        topic_id = max(doc_topics, key=lambda x: x[1])[0]
        summary = ldamodel.print_topic(topic_id)
        new_row = pd.Series({'document': filenames[i], 'topic_id':
topic_id, 'summary': summary})
        doc_topics_df = pd.concat([doc_topics_df,
pd.DataFrame([new_row])], ignore_index=True)
```

Once the topics have been identified, we display them in a pandas DataFrame[7] for easy interpretation of the documents and their respective topics:

```
pd.options.display.max_colwidth = 1000

doc_topics_df
```

Out[45]:

	document	topic_id	summary
0	Request+for+Proposal+Data+Software+Services+FA8806-23-R-0001+31+Jan+23.pdf	6	0.579*"software" + 0.156*"data" + 0.110*"cyber" + 0.043*"cybersecurity" + 0.029*"modeling" + 0.025*"compliance" + 0.022*"simulation" + 0.014*"automation" + 0.007*"devsecops" + 0.007*"jcids"
1	Request+for+Proposal+FA8806-23-R-0001+Data+Software+Services+17+Jan+2023.pdf	6	0.579*"software" + 0.156*"data" + 0.110*"cyber" + 0.043*"cybersecurity" + 0.029*"modeling" + 0.025*"compliance" + 0.022*"simulation" + 0.014*"automation" + 0.007*"devsecops" + 0.007*"jcids"
2	N6660420R8173 Science and Technology Broad Agency Announcement 20220914.pdf	0	0.651*"data" + 0.274*"software" + 0.042*"compliance" + 0.023*"cloud" + 0.009*"cyber" + 0.000*"modeling" + 0.000*"simulation" + 0.000*"visualization" + 0.000*"automation" + 0.000*"cybersecurity"
3	DD1423 (Solicitation) (AIML).doc	6	0.579*"software" + 0.156*"data" + 0.110*"cyber" + 0.043*"cybersecurity" + 0.029*"modeling" + 0.025*"compliance" + 0.022*"simulation" + 0.014*"automation" + 0.007*"devsecops" + 0.007*"jcids"
4	U_BIG_BAA_HM0476-16-BAA-0001_Amend6_Topic7_19Apr18.pdf	6	0.579*"software" + 0.156*"data" + 0.110*"cyber" + 0.043*"cybersecurity" + 0.029*"modeling" + 0.025*"compliance" + 0.022*"simulation" + 0.014*"automation" + 0.007*"devsecops" + 0.007*"jcids"
5	DARPA-SN-13-30_-_PPAML_Proposers'_Day_Announcement.pdf	6	0.579*"software" + 0.156*"data" + 0.110*"cyber" + 0.043*"cybersecurity" + 0.029*"modeling" + 0.025*"compliance" + 0.022*"simulation" + 0.014*"automation" + 0.007*"devsecops" + 0.007*"jcids"
6	Data Logging Market Survey.docx	0	0.651*"data" + 0.274*"software" + 0.042*"compliance" + 0.023*"cloud" + 0.009*"cyber" + 0.000*"modeling" + 0.000*"simulation" + 0.000*"visualization" + 0.000*"automation" + 0.000*"cybersecurity"

To drill it down even further, we can use a mask where we cast the values of the summary columnn to str and check to see if it contains a certain keyword, in this example - “data”:

```
mask = (doc_topics_df['topic_id'] == 0) &
(doc_topics_df['summary'].str.contains('data', case=False))
data_topics_df = doc_topics_df[mask]

data_topics_df
```

Out[46]:

	document	topic_id	summary
2	N6660420R8173 Science and Technology Broad Agency Announcement 20220914.pdf	0	0.651**data" + 0.274**software" + 0.042**compliance" + 0.023**cloud" + 0.009**cyber" + 0.000**modeling" + 0.000**simulation" + 0.000**visualization" + 0.000**automation" + 0.000**cybersecurity"
6	Data Logging Market Survey.docx	0	0.651**data" + 0.274**software" + 0.042**compliance" + 0.023**cloud" + 0.009**cyber" + 0.000**modeling" + 0.000**simulation" + 0.000**visualization" + 0.000**automation" + 0.000**cybersecurity"

Future Iterations

To continue improving on this idea, there are a multitude of avenues that we can take. In the topic modeling section, we're not taking advantage of our TF-IDF vectorizer. In order to gain more actionable insight from these documents, we plan to use either a graph algorithm such as TextRank[8] similar to Google's PageRank, or a large language model such as BART[9] (sequence-to-sequence model), T5[10] (text-to-text transformer model) or GPT-4[11] (dedicated language model).

Usage

If you want to try this out for yourself, clone the source code (https://github.com/INflow-Federal/contract-analysis) repository on our GitHub and open the jupyter notebook in your conda environment in terminal using `jupyter notebook contract-analysis.ipynb`. Any questions? Feel free to contact us and ask away.

1. https://github.com/conda-forge/miniforge [https://github.com/conda-forge/miniforge] ↩

2. https://docs.python.org/3/library/venv.html [https://docs.python.org/3/library/venv.html] ↩

3. https://pypi.org/project/PyPDF2/ [https://pypi.org/project/PyPDF2/] ↩

4. https://pypi.org/project/docx/ [https://pypi.org/project/docx/] ↩

5. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html [https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html] ↩

6. http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/ [http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/] ↩

7. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html [https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html] ↩

8. https://medium.com/analytics-vidhya/sentence-extraction-using-textrank-algorithm-7f5c8fd568cd [https://medium.com/analytics-vidhya/sentence-extraction-using-textrank-algorithm-7f5c8fd568cd] ↩

9. https://huggingface.co/facebook/bart-large [https://huggingface.co/facebook/bart-large] ↩

10. https://huggingface.co/docs/transformers/model_doc/t5 [https://huggingface.co/docs/transformers/model_doc/t5] ↩

11. https://openai.com/research/gpt-4 [https://openai.com/research/gpt-4] ↩