CS EE World
https://cseeworld.wixsite.com/home
May 2020
27/34
A

Submitter Info: Anonymous

An Application of Natural Language Processing

To what extent is the TF-IDF score a reliable means of classifying text and predicting job titles?

Subject: Computer Science

Word Count: 3991

Contents

1 Introduction

The ability of a computer to study and analyze language seems perplexing; when presented with text the system must be able to find three key pieces of information: the meanings of words, the way in which words are arranged in a sentence, and the context of the words. While this may seem familiar to a human, for a computer this poses a challenge. The three aforementioned types of information are known as semantic, syntax, and context information respectively, and combined they form the basis of natural language processing (Barba).

Natural language processing (NLP) is essentially the application of various algorithms and statistical models with the aim of analyzing language and making conclusions. Two key domains of NLP are information retrieval and text classification; the former is the ability to extract information from text, while the latter is the ability to classify text into distinct categories. Statistical models for NLP have been applied for several decades, but the advent of new technology has allowed these models to be applied on a large scale. With the rise of the Internet came an exponential growth in online publications, but such publications would be rendered pointless if they could not be retrieved in an efficient manner. Although most of the algorithms used nowadays are relatively simple in nature, such simplicity is what makes them powerful. These algorithms are in fact used by some of the most well-known search engines such as Google to match Web pages to a search query based on their relevance (Manning et al. xxxi-xxxii).

Before any string similarity algorithms can be applied to text, the text must be converted into a numerical format that a computer can easily manipulate. One way of accomplishing this is to count the number of times a word occurs in a piece of text (a document); this is known as the raw term frequency of the word. The problem with raw term frequency is that it only applies on the level of one document, and it does not account for the relevance of a word compared to other documents in the collection. Such a collection is referred to

as the *corpus*, and the set of all words that appear in the corpus at least once is known as the *dictionary*. The issues faced with raw term frequency can be rectified by instead quantifying words using a numerical statistic known as the TF-IDF (Term Frequency, Inverse Document Frequency) score, which favors words that are not only common in documents but can also be used to uniquely identify specific documents (Manning et al. 117-119).

This paper is a comparative study that answers the research question of: “To what extent is the TF-IDF score a reliable means of classifying text and predicting job titles?” To investigate this, two different string similarity algorithms were applied to a use case of matching job descriptions to a universal set of job titles. Each algorithm was run twice - once using raw term frequencies and once using TF-IDF scores. The goal of this study is to identify how much stronger of an indicator the TF-IDF score is as a way to quantify words in a corpus, compared to raw term frequency.

2 Background

2.1 The Situation

The data for this study was collected from an employment search company which faces the issue of ambiguity and/or vagueness with respect to job titles. This is because of a lack of international standards for job titles coupled with the fact that employers may provide a generic title. For example, titles such as “engineer” and “manager” do not provide enough detail for a job seeker. Therefore, a machine learning and NLP-based solution was implemented to analyze job descriptions and predict the best-matching job title. There is a wide range of algorithms that can be used to solve this problem, two of which will be explored in this paper: the weighted sum of scores, and the cosine similarity algorithm.

2.2 Calculating Raw Term Frequency

The raw term frequency of a word is simply the number of occurrences of a word in the document of interest. It does not account for document length, meaning that a word occurring 10 times in a 100-word document will have the same raw term frequency score as a word occurring 10 times in a 10,000-word document. This is one of the main flaws with raw term frequency. Nonetheless, since more relevant words are likely to be more common, raw term frequency is still a reliable way to weigh words based on their relevance (Jain).

Raw term frequency uses what is known as the *bag of words* approach of information retrieval. With this approach, the order of the words in a document is ignored (Brownlee). For example, although the sentences “Alice is taller than Bob” and “Bob is taller than Alice” have opposite meanings, they will be considered identical with the bag of words approach.

The raw term frequency of a word x in a document d is usually denoted as f_{xd}.

2.3 Calculating TF-IDF

TF-IDF is a numerical statistic that applies some of the shortcomings of raw term frequency to develop a more refined and meaningful indicator. TF-IDF is maximized for words that occur frequently in any one document but can only be found in that document and not any others. Therefore, words with higher TF-IDF scores for a certain document can be used to uniquely identify that document. This scoring system also uses the bag of words approach, since it treats each word individually and not as part of a sentence. The TF-IDF score of a word is determined by two parameters, the *term frequency* (tf) and the *inverse document frequency* (idf), which are multiplied together to result in a TF-IDF value (Jain).

Similarly to raw term frequency, the tf parameter is a measure of how common a word is in any given document. Each word in the dictionary has a different tf value for each document. The experiments in this study utilized the *sublinear* tf, which rescales the raw term frequency values on a logarithmic scale. This is in order to mitigate the effects of highly recurring words that are not necessarily relevant. The sublinear tf of a word x in document d is given by the following:

$$tf_{xd} = 1 + ln(f_{xd})$$

where f_{xd} is the raw term frequency as described previously.

Certain words may appear very frequently within one document, but that alone does not provide enough context as to how relevant the words are with respect to the entire corpus. Therefore, another parameter must be considered, which is the inverse document frequency. This parameter is built on the premise that a word that is unique to one document strongly signifies that document, but words that occur in many documents cannot be used to differentiate the documents (Manning et al. 118-119). Each word in the dictionary only has one idf value, which accounts for all of the documents in the corpus.

The idf of a word x is given by the following:

$$idf_x = 1 + ln\frac{1+N}{1+n}$$

where N is the total number of documents in the corpus, and n is the number of documents in which the word x can be found at least once. There are many variations of this formula; the formula given here is what the SciKit-Learn library uses in its TF-IDF calculations and what is used for the rest of this paper.

With this information, the TF-IDF score for a word in a document can be obtained. It is the product of the tf and idf values for that word and document pairing.

2.4 Generating a Document-Term Matrix

Once the raw term frequencies or TF-IDF scores are calculated, they can be inputted into what is known as a *document-term matrix*. Not only does this allow for easy visualization of the scores, but it is also necessary for the classification algorithms to work (Bishop). Figure 2-1 below is an example of a document-term matrix, in which the rows (A-E) indicate words and the columns (X, Y, and Z) indicate documents.

Word	X	Y	Z
A	5	4	3
B	2	6	1
C	1	8	7
D	1	9	6
E	10	2	5

Figure 2-1: A document-term matrix with hypothetical scores (The candidate).

2.5 The Weighted Sum

The weighted sum algorithm adds up the scores of the desired words for each of the documents, and chooses the document with the highest sum as the nearest match. This is an efficient method to solve the problem because job descriptions contain technical terminology relevant to the job title and finding those words is necessary. The weighted sum algorithm will favor those words if they occur in the query, especially if they occur more often.

Using Figure 2-1 as an example, suppose a query contains word A twice, C once, and D once. The matching scores for those words can be added to give a weighted sum, while also multiplying words that occur more frequently in the query. In this example, the weighted sum for document X would be:

$$(5 \times 2) + 1 + 1 = 12$$

the weighted sum for document Y:

$$(4 \times 2) + 8 + 9 = 25$$

and for document Z:

$$(3 \times 2) + 7 + 6 = 19$$

When comparing the three sums, it can be seen that document Y has the greatest sum, therefore the algorithm would predict document Y as the nearest match to the query.

2.6 Cosine Similarity

Another method to calculate string similarity is with a vector-based approach, wherein documents are represented in a vector space. Each column from the document-term

matrix is converted to a vector, with the number of dimensions equal to the number of words in the dictionary. These vectors are known as the document vectors (Manning et al. 120-124). As an example, the document vector for document X in Figure 2-1 is:

$$\begin{pmatrix} 5 \\ 2 \\ 1 \\ 1 \\ 10 \end{pmatrix}$$

Since there are 5 words in the dictionary, the vector is 5-dimensional, with each dimension corresponding to the score of a word for that document.

When a query is processed by the algorithm, that query's vector is plotted in the vector space in the same way that the document vectors are plotted. The similarity between the query vector and any of the document vectors can be determined by considering the angle that is made between the query vector and the document vector in question. The cosine of this angle will range from 0 to 1, with 1 corresponding to exactly the same vector and 0 corresponding to an exactly orthogonal (perpendicular) vector. The smaller the angle, the higher the cosine value and the more similar the vectors are. The cosine of an angle θ between two vectors $\vec{u}$ and $\vec{v}$ is given by the following:

$$cos\theta = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \times \|\vec{v}\|}$$

The numerator of the fraction denotes the dot product of the vectors, while the denominator denotes the product of the magnitudes (lengths) of the two vectors. The dot product is found by multiplying the matching components of both vectors and adding up the results. The magnitude of a vector is found by adding up the squares of all of the components of the vector and taking the square root of the result (Gupta).

An example of cosine similarity using simple 2-dimensional vectors is as follows:

$$\vec{u} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \vec{v} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

$$\vec{u} \cdot \vec{v} = (1 \times 4) + (3 \times 2) = 10$$

$$\|\vec{u}\| = \sqrt{1^2 + 3^2} = \sqrt{10}$$

$$\|\vec{v}\| = \sqrt{4^2 + 2^2} = \sqrt{20}$$

$$\therefore cos\theta = \frac{10}{\sqrt{10}\sqrt{20}} \approx 0.707$$

Therefore, the two vectors have a cosine similarity of about 0.707. Once the cosine similarity between the query vector and each of the document vectors is calculated, the document vector that has the highest cosine similarity with the query vector is what the algorithm predicts as the nearest match.

The cosine similarity method was used because a vector is an efficient way to represent all of the scores in one mathematical structure. Also, the angle between two vectors is constant irrespective of their length. This can be proven mathematically as follows: in order to double the length of one of the vectors, all of its components must be doubled, which in turn would result in the dot product being doubled. Since the dot product and the product of the lengths have both increased by the same scale factor, the angle is unchanged (Emmery).

3 Methodology

3.1 Data Initialization

The data set used for all of the experiments was a list of 7,788 records. The job descriptions acted as the independent variable and the corresponding job title acted as the dependent variable. There were 17 different possibilities of job titles; however, there was not a perfectly even distribution of records across the 17 titles. This is not a major problem, provided that the data is randomly chosen.

The data was represented using the DataFrame data structure from the Pandas library. This is a data structure that displays data in a table format and can transform data from a comma-separated values (CSV) file. Moreover, representing the data in a DataFrame allows for ease of manipulation and visualization (Willems). Figure 3-1 shows an example of a DataFrame.

	job_title	desc
0	sales manager	Do you have a successful background in sales, ...
1	area manager	The Regional Manager is responsible for creati...
2	area manager	The successful candidate will professionally p...
3	business development manager	Join a leading and award winning provider of i...
4	accountant	We look forward to your application! When appl...

Figure 3-1: The DataFrame for this data set, showing the first 5 rows (The candidate).

Once the data set was initialized, it was split into training and test data. Training data is data that the model uses to learn, and test data is data that the model has not seen before and is used to test the accuracy of the model (Venkatesh). It was decided that the test data constituted 30% of the original data set, and all four experiments used the same training and test data split for consistency purposes. This split resulted in 5,451 records for training data and 2,337 records for test data.

The training corpus was modified to consist of 17 large documents as opposed to the 5,451 individual records in the training set. Each of those new documents was an aggregate of all of the records in the training set that matched a certain job title. This was done because the ultimate goal of the study was to classify new documents according to their nearest match to a job title, rather than their nearest match to an individual job description.

3.2 Preprocessing

Before the documents could be manipulated, they had to be cleaned up; this is known as preprocessing. First, all of the text was converted to lowercase. Then, all punctuation, numbers, new lines, and extraneous whitespace were removed using RegEx (Regular Expression) patterns. These patterns filter out characters that should be removed from the documents. A special function called “text_cleaner” was created to apply all of the RegEx patterns. Next, all words from a list of stopwords were removed. Stopwords are common words that provide little to no meaning and are only present for grammatical purposes, such as “the,” “a,” and “is.” Finally, the document was converted to a list with each element being a word, as opposed to a single string. This is for ease of counting words (Jain). After these preprocessing steps were complete, the data processing could begin. Figure 3-2 shows an example of how a document can be preprocessed.

Original text:
```
'We are searching for candidates who desire
a challenging career and want to be part
of a high energy, supportive and rewarding
work environment.'
```

Text after preprocessing:
```
['searching', 'candidates', 'desire',
 'challenging', 'career', 'high', 'energy',
 'supportive', 'rewarding', 'work',
 'environment']
```

Figure 3-2: A sample document before and after preprocessing (The candidate).

3.3 The Experiments

Each algorithm (weighted sum of scores, and cosine similarity) was run once using raw term frequencies and once using TF-IDF scores, resulting in a total of four experiments.

The TF-IDF scores were calculated using the TfidfVectorizer from the SciKit-Learn library. For the experiments involving raw term frequency, a CountVectorizer was used instead. The CountVectorizer works in a similar way to the TfidfVectorizer, except that it produces the raw term frequencies of words rather than the TF-IDF scores. These two Vectorizer objects contain functions that were applied to create document-term matrices, which also incorporated the DataFrame data structure (Maklin). As an example, Figure 3-3 shows the first 5 rows of the raw term frequency document-term matrix, sorted in descending order for the job title of accountant.

	accountant	area manager	assistant store manager	business analyst	business development manager	contracts manager	electrical engineer	engineers	finance manager	general manager
accounting	2179	50	6	64	64	3	12	10	1094	63
financial	1850	73	470	257	215	75	19	42	1662	473
experience	1450	775	1351	1041	1175	604	1080	1371	1400	828
company	1208	576	644	367	805	366	285	515	1034	673
monthly	1035	123	47	122	274	142	37	154	707	156

Figure 3-3: Part of the raw term frequency document-term matrix (The candidate).

After the document-term matrices were generated, the application of the algorithms could begin. Each job description in the test data was processed by the algorithms to obtain a prediction of the nearest matching job title. This prediction was then compared with the actual job title for the job description to check whether or not the model predicted correctly. At the end of each experiment, the program outputted a percentage accuracy score, indicating the percentage of job descriptions that were correctly predicted. This percentage was compared to a threshold value of 80%; this is generally seen as the threshold above which an algorithm can be considered reliable for a particular use case.

When each experiment was completed, the program generated a CSV file consisting of the actual job title for each record in the testing data set coupled with what the model predicted. CSV files can be opened with Microsoft Excel, as seen in Figure 3-4:

	A	B	C	D
1		actual	predicted	outcome
2	0	area manager	general manager	FALSE
3	1	electrical engineer	electrical engineer	TRUE
4	2	business analyst	business analyst	TRUE
5	3	marketing manager	marketing manager	TRUE
6	4	accountant	accountant	TRUE
7	5	electrical engineer	software engineer	FALSE
8	6	area manager	area manager	TRUE
9	7	accountant	accountant	TRUE
10	8	sales manager	sales manager	TRUE
11	9	human resources executive	human resources executive	TRUE

Figure 3-4: A sample CSV output, showing the first 10 rows (The candidate).

Moreover, a *confusion matrix* was generated for each of the four experiments. A confusion matrix is a chart that shows how well the model predicted the dependent variable (in this case, job title) compared to what the actual values are (Manning et al. 307-308). As the name suggests, these matrices provide insight into the most frequently confused job titles.

All code was written in the Python language and can be found in the appendix.

4 Results

Table 4-1 shows the overall accuracy score for each of the four experiments.

Experiment	Overall Accuracy Score
Experiment 1: Sum of Raw Term Frequencies	51.78%
Experiment 2: Sum of TF-IDF Scores	84.72%
Experiment 3: Cosine Similarity with Raw Term Frequency	68.72%
Experiment 4: Cosine Similarity with TF-IDF	82.76%

Table 4-1: Accuracy scores for each experiment (The candidate).

Table 4-2 shows the accuracy scores for each job title for each of the four experiments, and an average. It provides insight into which job titles were predicted more accurately across the experiments.

Title	Experiment 1: Sum of Raw Term Frequencies	Experiment 2: Sum of TF-IDF Scores	Experiment 3: Cosine Similarity with Raw Term Frequency	Experiment 4: Cosine Similarity with TF-IDF	Average Accuracy Score
Assistant store manager	99%	99%	100%	100%	99.50%
Accountant	95%	95%	84%	91%	91.25%
Finance manager	53%	96%	100%	99%	87.00%
Software engineer	100%	99%	41%	81%	80.25%
Marketing manager	80%	88%	62%	77%	76.75%
Human resources executive	99%	97%	40%	64%	75.00%
Contracts manager	0%	98%	100%	99%	74.25%
Human resources manager	10%	87%	94%	97%	72.00%
Sales manager	98%	95%	39%	53%	71.25%
Business development manager	30%	79%	86%	89%	71.00%
Restaurant manager	17%	65%	99%	98%	69.75%
Business analyst	1%	60%	91%	96%	62.00%
Operations manager	0%	43%	91%	97%	57.75%
Engineers	59%	92%	13%	62%	56.50%
Electrical engineer	1%	46%	81%	95%	55.75%
Area manager	0%	67%	61%	77%	51.25%
General manager	4%	75%	36%	61%	44.00%

Table 4-2: Accuracy scores for each experiment, by job title (The candidate).

Table 4-3 shows how many unique job descriptions the data set had for each job title.

Title	# of records (in both training and test data sets)
Accountant	665
Software engineer	620
Sales manager	597
Assistant store manager	590
Engineers	575
Finance manager	541
Business development manager	503
Human resources manager	494
Human resources executive	479
General manager	423
Area manager	418
Marketing manager	410
Electrical engineer	360
Business analyst	347
Contracts manager	266
Restaurant manager	264
Operations manager	236

Table 4-3: Total number of records in the data set per job title (The candidate).

The following are the confusion matrices that were generated for each experiment:

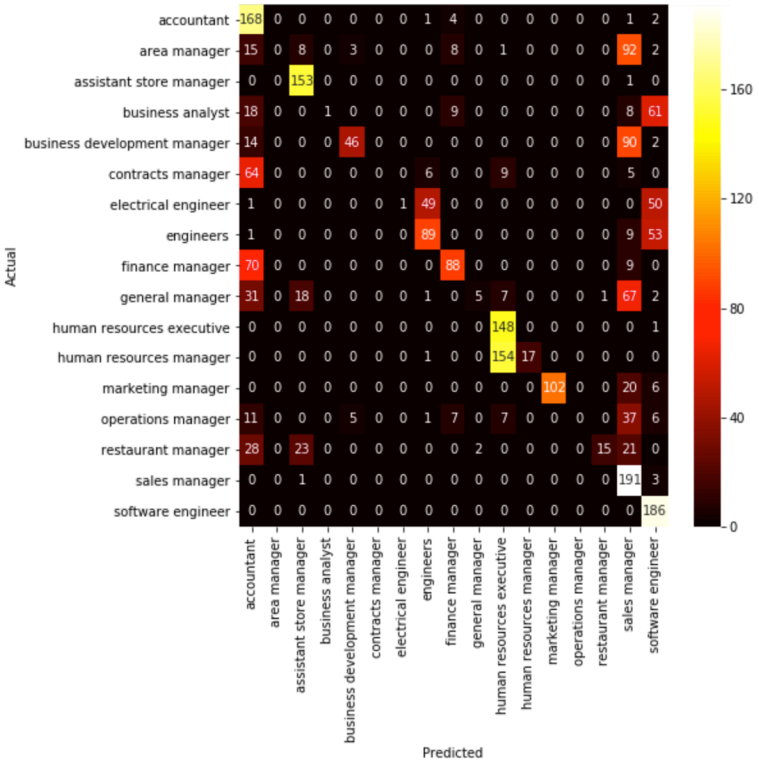

Figure 4-1: Confusion matrix for sum of raw term frequencies (The candidate).

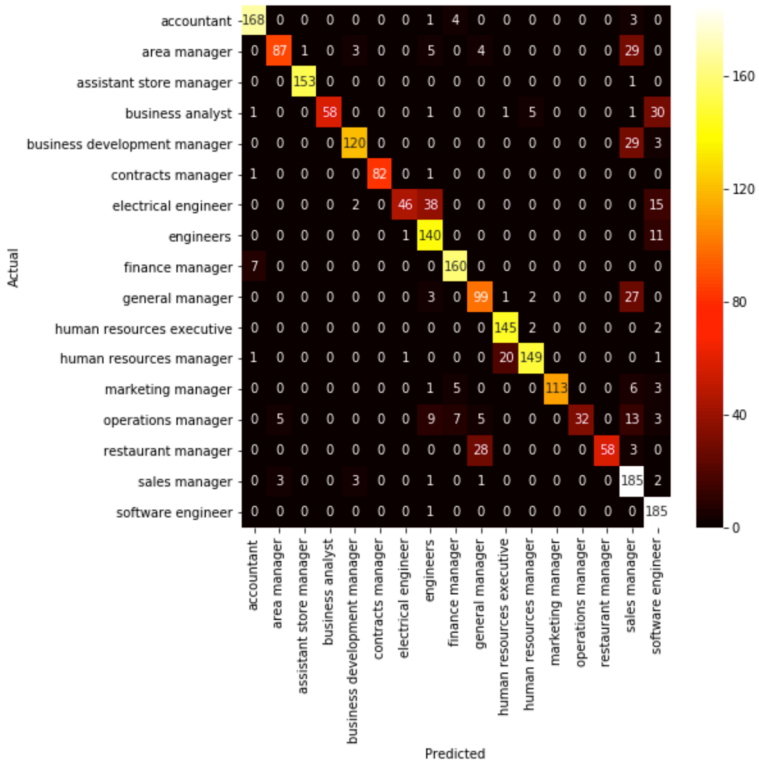

Figure 4-2: Confusion matrix for sum of TF-IDF scores (The candidate).

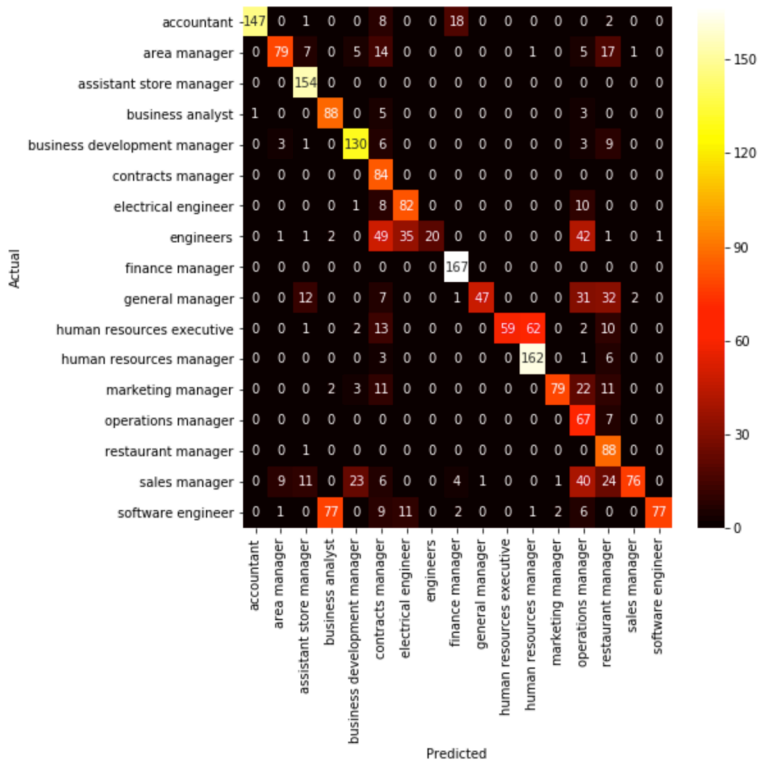

Figure 4-3: Confusion matrix for cosine similarity with raw term frequency (The candidate).

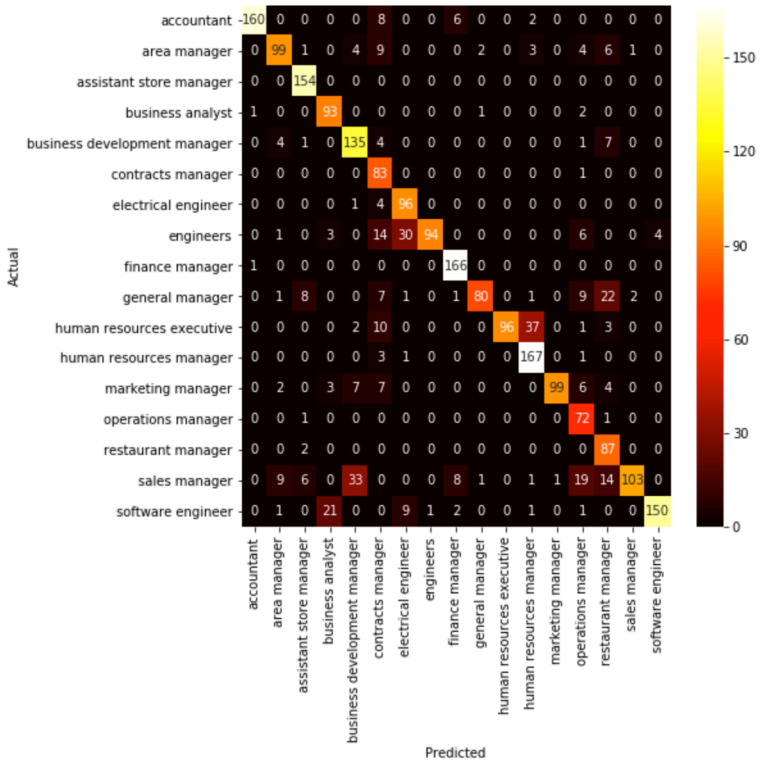

Figure 4-4: Confusion matrix for cosine similarity with TF-IDF (The candidate).

5 Analysis

Overall, it can clearly be seen from Table 4-1 that using the TF-IDF score had a pronounced effect on the accuracy of the algorithms, therefore making using the TF-IDF scores a more viable option than simply using raw term frequency. Both of the experiments that utilized TF-IDF scores had an accuracy score above 80%, which exceeds the threshold that was set prior to conducting the experiment. The accuracy scores generated from the experiments that utilized raw term frequency were notably lower, not only failing to reach the threshold of 80% but also falling severely below that.

The worst performer out of the four experiments was the sum of raw term frequency scores, with Table 4-1 indicating that it had an overall accuracy of only 51.78%. This is most likely due to the fact that this model can be heavily skewed by large numbers and that the raw term frequency values do not account for the lengths of the documents. This is especially a problem when considering that some job titles had more records than others, and therefore the job titles with the most records tended to contain higher raw term frequency values, therefore skewing the algorithm in their favor. This skewing is reflected in Table 4-2, with the four most frequently occurring job titles according to Table 4-3 (accountant, software engineer, sales manager, and assistant store manager) all having accuracy scores of at least 95% for this experiment. This can also be seen at the other end of the spectrum, with most of the least frequently occurring titles having alarmingly low accuracy scores. The titles of operations manager and contracts manager - the least and third-least frequently occurring titles respectively - had no correct predictions for this experiment. The high variability in accuracy for this experiment is more evident when considering its confusion matrix (Figure 4-1). Ideally, a confusion matrix should have a colored diagonal line from the top-left corner to the bottom-right corner, reflecting a majority of correct predictions. However, the confusion matrix for this experiment fails to produce a complete diagonal, and its unevenly scattered colored cells indicate many incorrect predictions.

Out of the 17 job titles, the most anomalous titles were those involving engineering - software engineer, electrical engineer, and the generic “engineers” title. The latter was the most problematic due to the fact that most of the records under the generic “engineers” title could potentially be applied to a software or electrical engineer depending on what words they contain. For example, in Figure 4-4, it can be seen that 30 records from the generic “engineers” title were predicted as an electrical engineer, which is likely due to the fact that those 30 records contained words that pertain to an electrical engineer. The issue is exemplified by the fact that in Table 4-2, the titles of electrical engineer and generic engineers had low average accuracy scores of 55.75% and 56.50% respectively. The presence of the generic “engineers” title seems to be a flaw with the data set, and it is plausible that repeating the experiments without the records with the generic “engineers” title would yield better accuracy scores for the algorithms.

However, it is worth noting that there were many words for software engineers that were not common for the generic “engineers” title, such as names of programming languages. Also, as seen in Table 4-3, there were more records for the title of software engineer (620) than there were for the other two titles involving engineering (575 and 360). With a greater sample size, the algorithms were able to learn more about words that are unique to software engineers. These were key reasons why the title of software engineer had a substantially higher accuracy score than the title of electrical engineer, at 80.25%.

Another common area of confusion for all four experiments was differentiating between a human resources executive and a human resources manager. For example, in Figure 4-2 it can be seen that 20 managers were predicted as executives, and in Figure 4-3 it can be seen that 62 executives were predicted as managers. This is understandable as both jobs are similar in nature and are therefore likely to have similar job descriptions, however a manager tends to have greater responsibility than an executive. This is not something that an algorithm with the bag of words approach would have been able to notice.

6 Conclusion

This study set out to answer the research question of: “To what extent is the TF-IDF score a reliable means of classifying text and predicting job titles?” When considering this question, it becomes more evident that using a bag of words approach such as TF-IDF scores was necessary. The overall goal of describing a word using such a score is to quantify the word according to its relevance among a set of documents, and the words with the highest scores tend to be words that can uniquely define a document/category (Manning et al. 119). When a human analyzes a job description and tries to match it to a job title, it is of their best interest to identify such keywords that can be used to clearly describe a job title, and by using TF-IDF scores to quantify words, the algorithms explored in this paper did essentially that.

Although an algorithm that produces an accuracy score of at least 80% is likely to be reliable for the specific use case, it is not to say that this study was perfectly executed. One area that this study could have been improved was in the sense that only *unigrams* (single words) were considered as part of the dictionary. This is an issue because there were many terms in the data set that constituted multiple words, such as “real estate” and “human resources.” Such *bigrams* (two-word phrases) should have also been included among the dictionary and their respective TF-IDF scores calculated. In this situation, all unigrams would have been treated as normal, but then every instance of two words adjacent to each other would have also been considered as a term (Kim).

Furthermore, the preprocessing of text could have been improved by applying a function to the text known as *lemmatization*. This function converts every word to its root form; for example, the words “manager” and “management” would be converted to “manage.” This is done because different forms of words are only used for grammatical purposes and thus do not provide any extra information to scoring systems using the bag of words approach. It also reduces the size of the dictionary, speeding up the process as a whole (Jain).

It can be seen that the bag of words approach is limited in the sense that it only considers the relevance of words relative to each other. However, there exist some algorithms that quantify words on an individual level based on the meaning of the word. In essence, these algorithms quantify words based on their innate features, and the grammatical order of words becomes more important (Nicholson). Some features that can be used to discern job titles include the field of work, annual salary, and level of education required. Therefore, words would be represented as vectors consisting of scores for all of those features. This method would likely solve the issue that was faced with distinguishing between a human resources manager with a human resources executive. However, implementing this is a convoluted process and was considered beyond the scope of this study.

Works Cited

Barba, Paul. “Machine Learning for Natural Language Processing.” *Lexalytics*, 25 Mar. 2019, www.lexalytics.com/lexablog/machine-learning-vs-natural-language-processing-part-1.

Bishop, Darrin. “Text Analytics - Document Term Matrix.” *Darrin Bishop*, 2 Oct. 2017, www.darrinbishop.com/blog/2017/10/text-analytics-document-term-matrix/.

Brownlee, Jason. “A Gentle Introduction to the Bag-of-Words Model.” *Machine Learning Mastery*, 9 Oct. 2017, machinelearningmastery.com/gentle-introduction-bag-words-model/.

Emmery, Chris. “Euclidean vs. Cosine Distance.” *Chris Emmery*, 25 Mar. 2017, cmry.github.io/notes/euclidean-v-cosine.

Gupta, Sanket. “Overview of Text Similarity Metrics in Python.” *Medium*, Towards Data Science, 16 May 2018, towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50.

Jain, Shubham. “Ultimate Guide to Deal with Text Data (Using Python) - for Data Scientists and Engineers.” *Analytics Vidhya*, 27 Feb. 2018, www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/.

Kim, Chan Woo. “Document Classification Part 2: Text Processing (N-Gram Model & TF-IDF Model).” *Medium*, Machine Learning Intuition, 23 Feb. 2018, medium.com/machine-learning-intuition/document-classification-part-2-text-processing-eaa26d16c719.

Maklin, Cory. “TF IDF: TFIDF Python Example.” *Medium*, Towards Data Science, 5 May 2019, towardsdatascience.com/natural-language-processing-feature-engineering-using-tf-idf-e8b9d00e7e76.

Manning, Christopher D., et al. *Introduction to Information Retrieval*. Cambridge University Press, 2009.

Nicholson, Chris. “A Beginner's Guide to Word2Vec and Neural Word Embeddings.” *Skymind*, skymind.ai/wiki/word2vec.

Venkatesh, Mothi. “What Is Training Data, Really?” *Hackernoon*, 19 Mar. 2018, hackernoon.com/what-is-training-data-really-adf0b97a116c.

Willems, Karlijn. “Pandas Tutorial: DataFrames in Python.” *DataCamp Community*, 14 Jan. 2019, www.datacamp.com/community/tutorials/pandas-tutorial-dataframe-python.

Appendix

text_cleaner function:

```
import re

pattern1 = re.compile(r'[\W+]')
pattern2 = re.compile(r'\b\d+\b')
pattern3 = re.compile(r' {2,}')

def text_cleaner(doc):
    doc = doc.replace('&', '  and ')
    doc = re.sub(pattern1, ' ', doc)
    doc = re.sub(pattern2, ' ', doc)
    doc = re.sub(pattern3, ' ', doc)
    return doc.lower().strip()
```

Main program:

```
# INITIALIZATION

import numpy as np
import pandas as pd
import re
import seaborn as sns

from text_cleaner import text_cleaner
from tqdm import tqdm

from nltk.corpus import stopwords
from pandas.api.types import CategoricalDtype
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split

stop_words = stopwords.words('english') # defining stopwords

df = pd.read_csv('dataset.csv').dropna() # reading the CSV file
df['job_title_cln'] = df['job_title_cln'].astype("category")
X, y = df['desc'], df['job_title_cln'] # defining the independent and dependent variables
titles = list(y.cat.categories)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

df = df.reset_index(drop=True)
corpus = []
```

```
train = pd.DataFrame({"col1": X_train, "col2": y_train})
test = pd.DataFrame({"col1": X_test, "col2": y_test})

for i in tqdm(range(len(titles))): # building the training and test corpora
    main = train.loc[train['col2'] == titles[i]]
    main = main.reset_index(drop=True)
    toAdd = ""
    for x in range(len(main)): # preprocessing of text using regex and removal of stopwords
        temp = re.split(" ", text_cleaner(main.loc[x, "col1"]))
        temp = [word for word in temp if word not in stop_words]
        toAdd += " ".join(temp)

    corpus.append(toAdd)

print("Intialization Complete.")

# RAW TERM FREQUENCY MATRIX GENERATOR

cv = CountVectorizer()
vector = cv.fit_transform(corpus)

feature_names = cv.get_feature_names()
final = pd.DataFrame(vector[0].T.todense(), index=feature_names, columns=[titles[0]])
for i in range(1, len(titles)):
    tempvec = vector[i]
    tempdf = pd.DataFrame(tempvec.T.todense(), index=feature_names, columns=[titles[i]])
    final = final.join(tempdf)

# TF-IDF DOCUMENT-TERM MATRIX GENERATOR

tv = TfidfVectorizer(norm=None, sublinear_tf=True)
vector2 = tv.fit_transform(corpus)

feature_names = tv.get_feature_names()
final2 = pd.DataFrame(vector2[0].T.todense(), index=feature_names, columns=[titles[0]])
for i in range(1, len(titles)):
    tempvec = vector2[i]
    tempdf = pd.DataFrame(tempvec.T.todense(), index=feature_names, columns=[titles[i]])
    final2 = final2.join(tempdf)

# FUNCTION TO CHECK SUMS

def check_scores(entry, data):
    query = re.split(" ", text_cleaner(entry))
    query = [word for word in query if word not in stop_words]
    generator = []
    for term in query:
```

```
        try:
            key = data.loc[term, :].tolist()
            generator.append(key)
        except KeyError:
            continue

    final_list = []

    for i in range(len(titles)):
        key2 = 0
        for j in range(len(generator)):
            key2 += generator[j][i]
        final_list.append(key2)

    return titles[final_list.index(max(final_list))]

# SUM CALCULATION

y_test = y_test.reset_index(drop=True)
test = test.reset_index(drop=True)
test_output = pd.DataFrame(columns=["actual", "predicted", "outcome"])
test_output2 = pd.DataFrame(columns=["actual", "predicted", "outcome"])

count = 0
count2 = 0

for x in tqdm(range(len(test))): # iterating through the test data
    check = check_scores(test.loc[x, "col1"], final)
    if check == y_test[x]:
        count += 1
        test_output.loc[x] = [y_test[x]] + [check] + [True]
    else:
        test_output.loc[x] = [y_test[x]] + [check] + [False]

    check = check_scores(test.loc[x, "col1"], final2)
    if check == test.loc[x, "col2"]:
        count2 += 1
        test_output2.loc[x] = [y_test[x]] + [check] + [True]
    else:
        test_output2.loc[x] = [y_test[x]] + [check] + [False]

test_output.to_csv("raw_sum_results.csv")
print("Raw Term Frequency Sum accuracy = " + str(count / len(test) * 100) + "%")
test_output2.to_csv("tfidf_sum_results.csv")
print("TF-IDF Weighted Sum accuracy = " + str(count2 / len(test) * 100) + "%")

# RAW TERM FREQUENCY WEIGHTED SUM CONFUSION MATRIX
```

```
matrix = confusion_matrix(test_output['actual'], test_output['predicted'])
fig, ax = plt.subplots(figsize=(7.5, 7.5))
sns.heatmap(matrix, annot=True, xticklabels=titles, yticklabels=titles, cmap='hot', ax=ax)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
plt.savefig('mat1.png')

# TF-IDF WEIGHTED SUM CONFUSION MATRIX

matrix = confusion_matrix(test_output2['actual'], test_output2['predicted'])
fig, ax = plt.subplots(figsize=(7.5, 7.5))
sns.heatmap(matrix, annot=True, xticklabels=titles, yticklabels=titles, cmap='hot', ax=ax)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
plt.savefig('mat2.png')

# RAW TERM FREQUENCY COSINE SIMILARITY

X_test_cv = cv.transform(X_test)
test_df = pd.DataFrame(X_test_cv.T.todense(), index=cv.get_feature_names())
cos_df = pd.DataFrame(columns=["actual", "predicted", "outcome"])
count = 0

for i in tqdm(range(len(test))):
    query_vector = test_df[i]
    cosine_list = []
    for j in titles:
        doc_vector = final2[j]
        query_vector = np.array(query_vector).reshape(-1, 1).swapaxes(0, 1)
        doc_vector = np.array(doc_vector).reshape(-1, 1).swapaxes(0, 1)
        cosine_list.append(
            cosine_similarity(query_vector, doc_vector).max()
        )

    pred = titles[cosine_list.index(max(cosine_list))]
    if pred == y_test[i]:
        count += 1
        cos_df.loc[i] = [y_test[i]] + [pred] + [True]
    else:
        cos_df.loc[i] = [y_test[i]] + [pred] + [False]

cos_df.to_csv("raw_cos_results.csv")
print("Raw Term Frequency Cosine Similarity accuracy = " + str(count / len(test) * 100) + "%")
```

```
# TF-IDF COSINE SIMILARITY

X_test_tfidf = tv.transform(X_test)
test_df = pd.DataFrame(X_test_tfidf.T.todense(), index=tv.get_feature_names())
cos_df2 = pd.DataFrame(columns=["actual", "predicted", "outcome"])
count2 = 0

for i in tqdm(range(len(test))):
    query_vector = test_df[i]
    cosine_list = []
    for j in titles:
        doc_vector = final2[j]
        query_vector = np.array(query_vector).reshape(-1, 1).swapaxes(0, 1)
        doc_vector = np.array(doc_vector).reshape(-1, 1).swapaxes(0, 1)
        cosine_list.append(
            cosine_similarity(query_vector, doc_vector).max()
        )

    pred = titles[cosine_list.index(max(cosine_list))]
    if pred == y_test[i]:
        count2 += 1
        cos_df2.loc[i] = [y_test[i]] + [pred] + [True]
    else:
        cos_df2.loc[i] = [y_test[i]] + [pred] + [False]

cos_df2.to_csv("tfidf_cos_results.csv")
print("TF-IDF Cosine Similarity accuracy = " + str(count2 / len(test) * 100) + "%")

# RAW TERM FREQUENCY COSINE SIMILARITY CONFUSION MATRIX

matrix = confusion_matrix(cos_df['actual'], cos_df['predicted'])
fig, ax = plt.subplots(figsize=(7.5, 7.5))
sns.heatmap(matrix, annot=True, xticklabels=titles, yticklabels=titles, cmap='hot', ax=ax)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
plt.savefig('mat3.png')

# TF-IDF COSINE SIMILARITY CONFUSION MATRIX

matrix = confusion_matrix(cos_df2['actual'], cos_df2['predicted'])
fig, ax = plt.subplots(figsize=(7.5, 7.5))
sns.heatmap(matrix, annot=True, xticklabels=titles, yticklabels=titles, cmap='hot', ax=ax)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
plt.savefig('mat4.png')
```