1. **Deliverable 1 For this assignment compare the results when no partitions are used to those with various partition percentages, but show results for only three partition (default, and higher/lower training set) combinations of partition percentages.**

For this part of the assignment, we are looking at how well our dependent variable is represented when splitting the dataset into train/validate/test sub-datasets. I am not using SAS for this assignment, so I am utilizing the Python sklearn tool to split my dataset.

Dependent Variable Percentages:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Score** | **No Split** | **Default (50/25/25)** | | |
| **Train** | **Validate** | **Test** |
| 1 | 7.25% | 7.04% | 8.21% | 6.70% |
| 2 | 5.74% | 6.08% | 4.86% | 5.94% |
| 3 | 10.45% | 9.88% | 11.74% | 10.30% |
| 4 | 21.00% | 21.08% | 19.65% | 22.17% |
| 5 | 55.57% | 55.91% | 55.54% | 54.90% |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Score** | **No Split** | **Large Training Set (75/10/15)** | | |
| **Train** | **Validate** | **Test** |
| 1 | 7.25% | 7.01% | 8.54% | 7.56% |
| 2 | 5.74% | 5.7% | 5.58% | 6.06% |
| 3 | 10.45% | 10.45% | 10.25% | 10.56% |
| 4 | 21.00% | 21.03% | 19.42% | 21.9% |
| 5 | 55.57% | 55.81% | 56.21% | 53.93% |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Score** | **No Split** | **Small Training Set (15/10/75)** | | |
| **Train** | **Validate** | **Test** |
| 1 | 7.25% | 6.78% | 6.66% | 7.42% |
| 2 | 5.74% | 6.18% | 6.21% | 5.59% |
| 3 | 10.45% | 10.14% | 10.26% | 10.54% |
| 4 | 21.00% | 21.73% | 20.7% | 20.89% |
| 5 | 55.57% | 55.16% | 56.17% | 55.57% |

Looking at the three different splits, all three different split ratios appear to do a fairly good job approximating the actual distribution of the dependent variable, score. That is to say, for any given score on any of the three tables, the train/validate/test column is always within 1-1.5% of the distribution in the full dataset.

1. **Deliverable 2: Show multi-word association tables or diagrams. A typical tf-idf table is shown in the csv results file in this Module’s folder. Show a similar tf-idf table with columns for POS, name entities, importance measure(s), and some visualization or table of word associations.**

TF-IDF Table; Document 0:

This table is created from the first document in the set. Columns are defined as:

* **pos**: part of speech
* **freq**: frequency of term in this document
* **numdocs**: number of documents containing this term
* **tokens\_in\_doc**: total number of distinct tokens in this document. This column is not used for the other metrics, just an interesting number.
* **weighted\_tf**: term frequency weighted by the most frequent token in the document
* **idf**: inverse document frequency. Log2 of N over the numdocs column
* **tfidf**: freq column multiplied by the idf column
* **tfidf**\_weighted: weighted\_tf multiplied by the idf column

It is interesting to note that the highest rated tfidf term is actually only the third most important term once weighting the frequency by document length.

A screenshot of a cell phone

Description automatically generated

We can create this same matrix at the corpus level, where the columns are the tokens, and each document is a row. Each cell can be populated with that document’s tfidf value. It is worth noting there are a lot of zeros in this table. That is because most of these tokens are not present in all documents, but just a subset of them (and thus, the TF term is equal to zero).  
A screenshot of a cell phone

Description automatically generated

Visualization of word associations:

Python did not have a utility to do this, so wrote a script to draw a graph. Edge thickness is scaled to collocation score.

“fly”:  
A picture containing map, sky, skiing

Description automatically generated

“fall”:

A picture containing map, sky, skiing, red

Description automatically generated

1. **Deliverable 3: Explain how stemming can lead to more reliable feature estimates. How does the use of a single token, like the stem, lemmata, and one of the terms in each SynSet (a set of all the different words meaning the same thing in a context), enhance the reliability of identifying the important concepts in a document/corpus?**

Stemming and lemmatization leads to a more reliable feature set, as it functionally groups otherwise equivalent concepts into a single feature. Both stemming and lemmatization techniques are similar, reducing the feature set by essentially deconjugating words (stemming decomposes words into parts, whereas lemmatization converts words to their non-plural infinitive forms).

A good example on why this could be important is tense. Imagine a dataset that has two documents that should be categorized the same. However, one of the documents is primarily in past tense. In this case, without stemming and lemmatization, our model would fail to equate terms such as driving and purchasing in document A to terms drove and purchased in document B.

This concept is further extended by the use of a SynSet. Using synonyms allow the model to extract concepts related to the text, rather than simply the words in the text itself. A good example would be comparing/classifying document written by different authors. These documents might contain very similar information, however, the authors themselves may use very different language to describe the concepts. One author might write “The lawyer requested the judge for a reduced sentence”, while the other writes “The attorney asked the justice for a smaller punishment”. Using only stemming and lemmatization, our model will not find much connection between these two otherwise near identical concepts. In this case, a SynSet would allow us to convert these synonyms into single tokens that represent the idea.

1. **Deliverable 4 Briefly describe what each of the outputs show and how their content relates to the material in Chapter 2 of Weiss et al.**

The tf-idf table encompasses a lot of the ideas described in Weiss. First of all, for my table I employed POS tagging. By default, the python nltk I used actually tagged the tokens using a greatly expanded POS set, similar to the Penn Tree bank described in the book. However, for simplicity and to match the example table, I changed the tagging to the ‘universal’ tagset (noun, adj, adverb).

Next, the book mentioned the removal of stopwords. At first, I removed all stop words and punctuation from my token set. Looking back at the book, I shouldn’t remove punctuation at the very least, until after tagging; the tagger likely uses sentence structure in its tagging algorithm. Additionally, later on in the process, a phrase/word association algorithm should also use sentence structure; as described in Weiss, phrases cannot cross sentence boundaries.

I also noticed is the occasional failure of the tokenizer. As the book mentioned, trying to disambiguate periods that end sentences, and periods that are parts of words/abbreviations/numbers can be difficult programmatically. This is something I noticed in one of my tables. Note, the token ‘groups’ has a trailing period.  


Weiss does not touch much on details about phrase recognition, but the concept is very similar to the collocation/word association clouds above. In essence, we are looking for words that are commonly found in proximity of one another. A mutual information measure can then ‘score’ how strongly these a words are related.