TOOL

Create Matrices from Text

Many mathematical models require numeric inputs and outputs. In natural language processing (NLP), it is common to represent textual elements as vector and matrix structures so that you can apply mathematics, statistics, and machine learning to these elements.

Use the first part of this tool to help you remember several of the most common types of matrices you can create from a corpus. Use the second part of this tool to help you represent your own documents as matrices.

Part One: Types of Matrices That Represent Text

Word or phrase frequency

One of the simplest ways to convert text into numerical representation is to count the number of occurrences of objects, such as words or phrases. These counts allow you to compare documents based on common frequencies of shared words. For example, the word *w* may appear once in document d_1, five times in d_2, and seven times in d_3. From these counts, you can deduce that d_2 and d_3 share more similarities.

Document–term matrix (DTM)

One numerical representation of a collection of documents is a document–term matrix (DTM), which has rows corresponding to sentences (documents) and columns corresponding to constituent words (terms). The values in a DTM are counts of the words in documents, where the element *i, j* in the matrix is the count for the i^{th} document and j^{th} term. As an alternative to count-based DTMs, some DTMs might instead track presence (indicated by 1) and absence (indicated by 0) of a word in a document.

A DTM can be used to compare the similarity between documents in a corpus. This matrix is often highly sparse, however, since only a very small subset of words in vocabulary are used in any given sentence. Further, frequency- and count-based DTMs do not include

information on the relative commonness or rarity of words within and between documents. Thus, a DTM can have high counts for low-importance words that do not help us distinguish or compare documents. For example, the article "the" would have a high count in the DTM but is too frequent to help you compare documents.

Term frequency-inverse document frequency (TF-IDF) DTM

When you want to use a matrix to compare similarity among documents, it can be helpful to take information on how many documents a term is found in into account. In contrast to stopwords like the word "the," rare words can be very effective in helping you compare document similarity. For example, if you have several documents containing the word "diabetes" and several documents containing the word "NASDAQ," you can be reasonably confident which documents are in the group of medical documents versus financial documents. Yet extremely rare words that only occur in one document don't provide much information.

The TF–IDF DTM can provide a better overall picture of similarity of documents in a corpus by giving higher weights to rarer words across documents as well as frequent words in fewer documents. Words that are frequent across all documents receive lower weight.

A DTM can be converted into a TF-IDF by dividing the term frequency by the document frequency. This matrix can be used to automatically detect stopwords (by thresholding low weights across documents) and important words.

Part 2: Creating DTMs and TF-IDF DTMs in Python

Use the following code and steps to create DTMs and TF-IDFs.

Using Python with this tool

The portions of this tool with a gray background are code text you can use to complete the examples included in this tool. You can also modify them to use with your own data. In these examples:

- Commands are the lines of code that don't begin with a pound sign (#). Type these lines into Python to carry out the command.
- Commented text begins with one pound sign and explains what the code does.

Load the documents.

Here, load 15 sentences about language as the object `LsQuotes`:

```
LsQuotes=[
 "Learn a new language and get a new soul.",                                  # Czech Proverb
 "A mistake is to commit a misunderstanding.",                                # Bob Dylan
 "Language is to the mind more than light is to the eye.",                    # William Gibson
 "Knowledge of languages is the doorway to wisdom.",                          # Roger Bacon
 "To have another language is to possess a second soul.",                     # Charlemagne
 "Change your language and you change your thoughts.",                        # Karl Albrecht
 "A different language is a different vision of life.",                       # Federico Fellini
 "The limits of my language mean the limits of my world.",                    # Ludwig Wittgenstein
 "He who knows no foreign languages knows nothing of his own.",               # Johann Wolfgang von Goethe
 "Learning is a treasure that will follow its owner everywhere.",             # Chinese Proverb
 "You can never understand one language until you understand at least two.",  # Geoffrey Willans
 "One language sets you in a corridor for life. Two languages open every door along the way.",  # Frank Smith
 "Language is the blood of the soul into which thoughts run and out of which they grow.",       # Oliver Wendell Holmes
 "A special kind of beauty exists which is born in language, of language, and for language.",   # Gaston Bachelard
 "One should not aim at being possible to understand but at being impossible to misunderstand."  # Marcus Fabius Quintilian
 ]
```

Create a term frequency DTM.

To create a DTM based on term frequency, you can use the Scikit-Learn `CountVectorizer()` object.

```
# create object to count pre-processed words
cv = CountVectorizer(stop_words='english', lowercase=True)
# retrieve vocabulary in list
LsVocab = cv.get_feature_names()
# create document-term matrix (DTM)
DT_smatrix = cv.fit_transform(LsQuote)                                  # sparse format
DT_dmatrix = DT_smatrix.toarray()                                       # dense format
dfDT = pd.DataFrame(DT_dmatrix, index=LsQuote, columns=LsVocab)     # dataframe format
```

Create a TF-IDF DTM.

You examined two ways to create a TF-IDF in this course. One method is to use the computed count-based DTM and apply the TF-IDF weighting formula to convert the counts to meaningful weights.

```
# use TfidTransformer on the dense DTM
smTFIDF1 = TfidfTransformer(norm='l2', use_idf=True, smooth_idf=True).fit_transform(DT_smatrix) # fit and
transform
# convert to dataframe
dfTFIDF1 = pd.DataFrame(smTFIDF1.toarray(), index=LsQuotes, columns=LsVocab)
```

The other method is to call `TfidfVectorizer` directly, which will do both the word counting and weighing.

```
# use TfidfVectorizer directly without creating DTM
tv = TfidfVectorizer()
smTFIDF2 = tv.fit_transform(LsQuotes)
# convert to dataframe
dfTFIDF2 = pd.DataFrame(smTFIDF2.toarray(), columns=tv.get_feature_names())
```

Suppose you had a new quote that was not available during training (i.e., fitting) time. You can apply the fitted vectorizer to the new quote to refit the TF-IDF.

```
# refit TfidfVectorizer object with new quote
newQuote = 'this is a new quote about language'
tvS16 = tv.transform([newQuote]).toarray()
# convert to dataframe
dfTFIDF2 = pd.DataFrame(tvS16, columns=tv.get_feature_names())
```