Cheatsheets / **Intro to Language Models in Python**

codecademy

Word Embeddings

Vectors in NLP

In natural language processing, *vectors* are very important! They contain numerical information that indicates the magnitude of the pieces of data being represented.

The dimension, or size, of a vector can be manipulated to allow for more data to be stored in the vector. Above, three one-dimensional vectors are shown.

In Python, you can represent a vector with a *NumPy* array. The following creates a vector of **even** numbers from 2 to 10.

```
even_nums = np.array([2,4,6,8,10])
```

codecademy

Word Embeddings

Word embeddings are key to natural language processing. Each is a real number vector representation of a specific word. Contextual information about that word is encoded within the vector numbers.
A basic English word embedding model can be loaded in Python using the `spaCy` library. This allows access to embeddings for English words.

```
nlp = spacy.load('en')
```

Call the model with the desired word as an argument and access the `.vector` attribute:

```
nlp('peace').vector
```

The result would be:
`[5.2907305, -4.20267, 1.6989858, -1.422668, -`

Distance Between Vectors

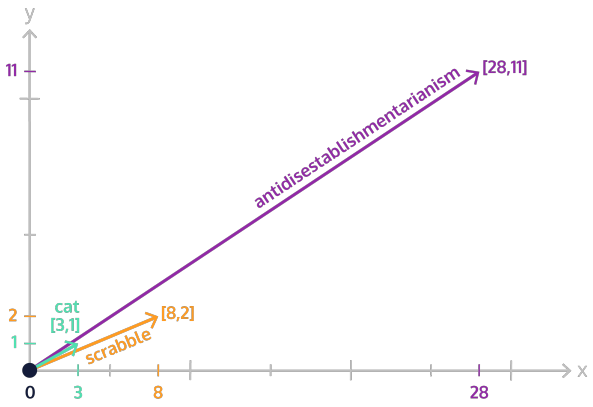
Measuring distances between word embedding vectors allows us to look at the similarities and differences between words. This type of distance can be calculated using either *Manhattan*, *Euclidean* or *Cosine distance*.
With large-dimensional vectors, **Cosine distance** is preferred because Manhattan and Euclidean distances can become too large. Cosine produces **much smaller** values; it measures the angle between two vectors, whereas the other two calculate distances between vector points.
The Python library *SciPy* contains functions to calculate each easily given two vectors `a` and `b`.

```
from scipy.spatial.distance import
cityblock, euclidean, cosine

manhattan_d = cityblock(a,b) # 129.62

euclidean_d = euclidean(a,b) # 16.45

cosine_d = cosine(a,b) # 0.25
```

codecademy

Word Contexts in Vector Space

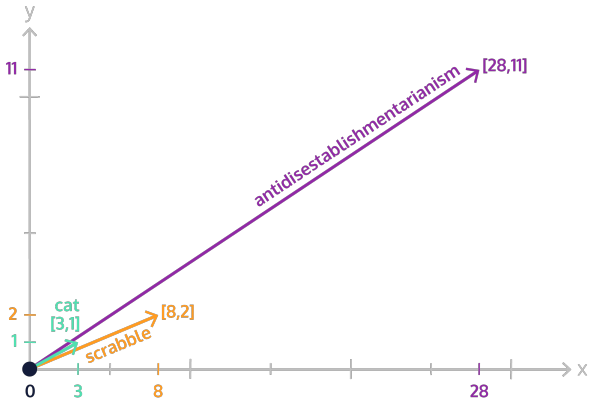
Words used in similar contexts have similar word embeddings, meaning that they are mapped to similar areas in the vector space.
Words in similar contexts have a **small** *cosine distance*, while words in different contexts have a **large** *cosine distance*.
The vectors for “cat” and “scrabble” point to similar areas of the vector space, and “antidisestablishmentarianism” points to a very different area.
“cat” and “scrabble” would have a **small** cosine distance, and “cat” and “antidisestablishmentarianism” would have a **large** one.



Word2Vec Algorithm

To compare word embedding vectors, vector values need to first be appropriately created!
Word2vec is a statistical learning algorithm that creates embeddings using written text (a *corpus*).
There are two different architectures of the corpus it can use:

- *Continuous Bag of Words*: the algorithm goes through each word in the training corpus, in order, and predicts the word at each position based on applying bag-of-words to surrounding words. The order of the words does **not** matter!
- *Continuous Skip-Grams*: Look at sequences of words that are separated by some specified distance, as opposed to the common practice of looking at groups of n-consecutive words in a text (*n-grams*). The order of context **is** taken into consideration!



codecademy

Gensim Embeddings Creation

Instead of using pre-trained word embeddings from *spaCy*, you can create your own unique ones! The Python library `gensim` allows a *word2vec* model to be trained on any corpus of text. These embeddings can be any specified dimension and are unique contextual embeddings to that corpus of text.

- `corpus` : A list of lists. Each inner list is a document in the corpus, each element in the inner lists is a word token
- `size` : The dimensions of the embeddings.
- Don't worry about the other arguments.

Model attributes:

- `.wv.vocab.items()` – vocabulary of the model
- `.most_similar("cat", topn=10)` – 10 most similar words to "cat"
- `.doesnt_match(["mouse", "cat", "europe"])` – identify the word **least** like others

Print Share

```
import gensim

model = gensim.models.Word2Vec(corpus,
size=100, window=5, min_count=1,
workers=2, sg=1)
```