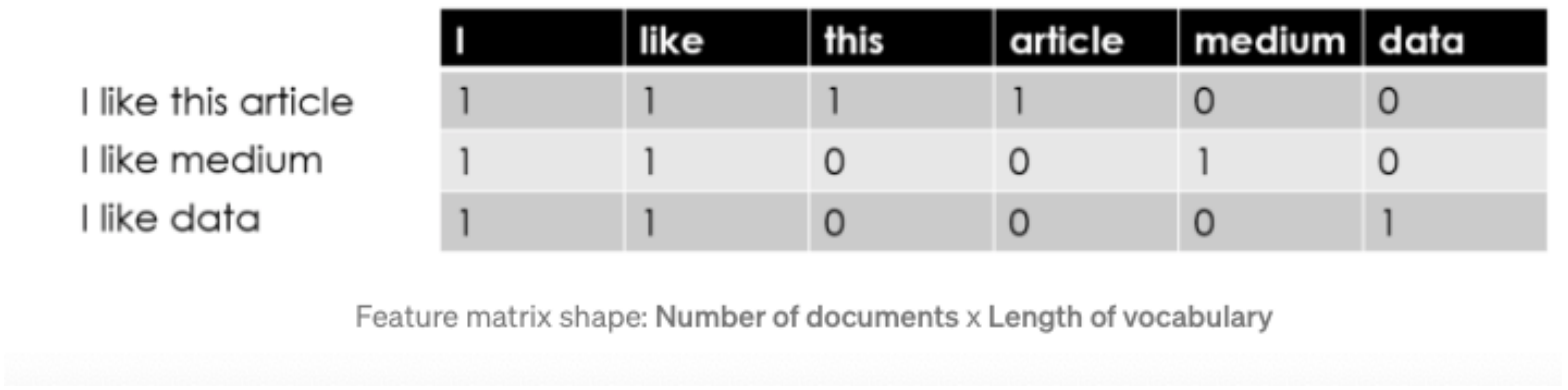
General Approaches to Understanding Content with NLP

Brooke Luetgert

Standard approaches

- NLTK- look at Twitter Capitol Hill data
- Word Embeddings- IMDB data
- Word2Vec
- Text Classification with an RNN
- Classify text with BERT

NLP tools- Bag of Words



	I	like	this	article	medium	data
I like this article	1	1	1	1	0	0
I like medium	1	1	0	0	1	0
I like data	1	1	0	0	0	1

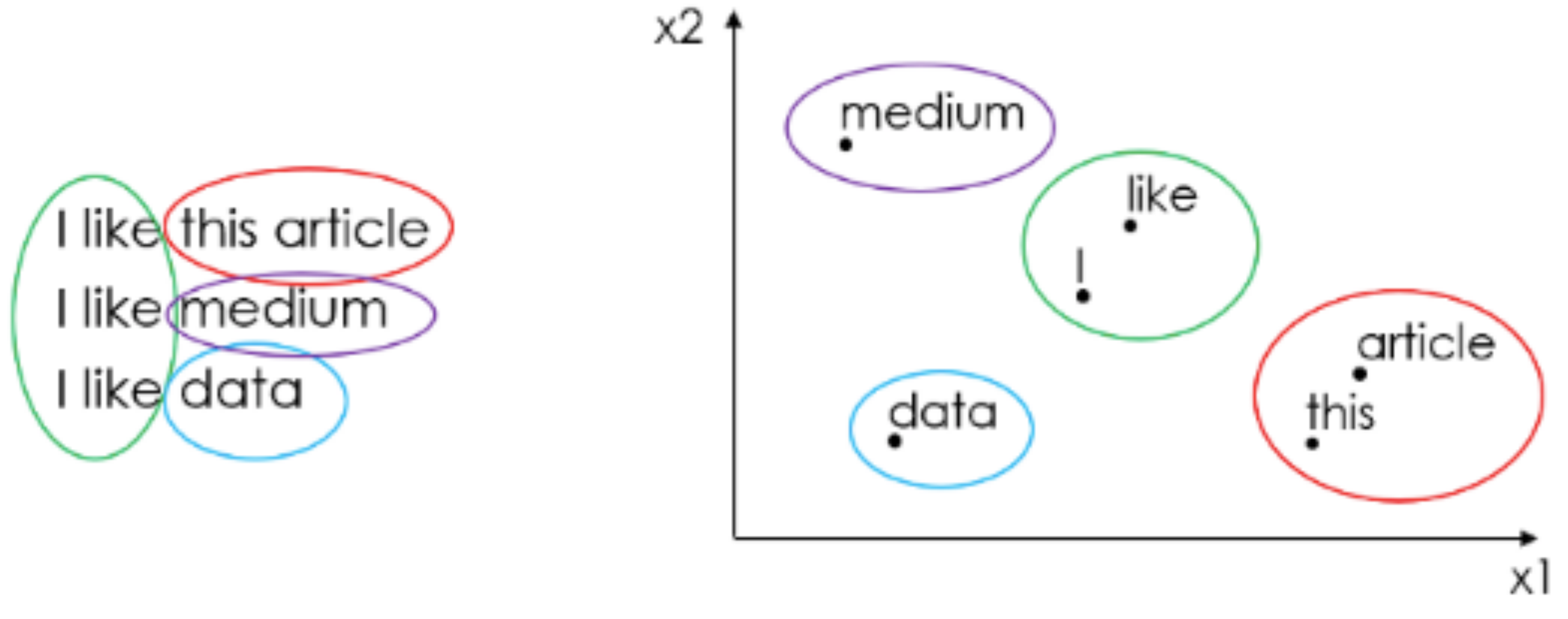
Feature matrix shape: **Number of documents** x **Length of vocabulary**

- **Bag of Words** (Tf-Idf)- simple ML, can be implemented with NLTK. Builds vocabulary from a corpus of documents and counts how many times words appear in each document. Each word in the vocabulary becomes a feature and a document is represented by a vector with the same length of vocabulary (This is the "bag of words")

- <u>Caution</u>: more documents mean larger vocabulary, creating huge sparse matrix. Text analysis based on frequency can provide a distorted view. Term-frequency, inverse document frequency (Tf-Idf) allows the value of a word to increase proportionally to count, but it is inversely proportional to the frequency of the word in the corpus

NLP tools- Word Embedding

- **Word embeddings**- NN don't like string data. Words need to be encoded to create vectors of information as numbers. (Similar to our transformation of images earlier) Vectors are calculated from the probability distribution for each word appearing before or after another. Words of the same context often appear together in a corpus, so they will be close in the vector space as well.
  - Word2Vec is a collection of related models used to create word embeddings. These are shallow 2-layer neural nets usually containing several hundred dimensions. Corpus size is crucial.



Words embedded in 2D vector space

NLP tools- Language Models

- **Language models** with feature engineering and transfer learning from pre-trained BERT. These are contextualized, dynamic word embedding. These models are designed to address polysemy disambiguation, that is the problem of a word having very different meanings (things “bank” or “stick”). These models look at the entire sentence and assign embedding relative to the sentence for each word.

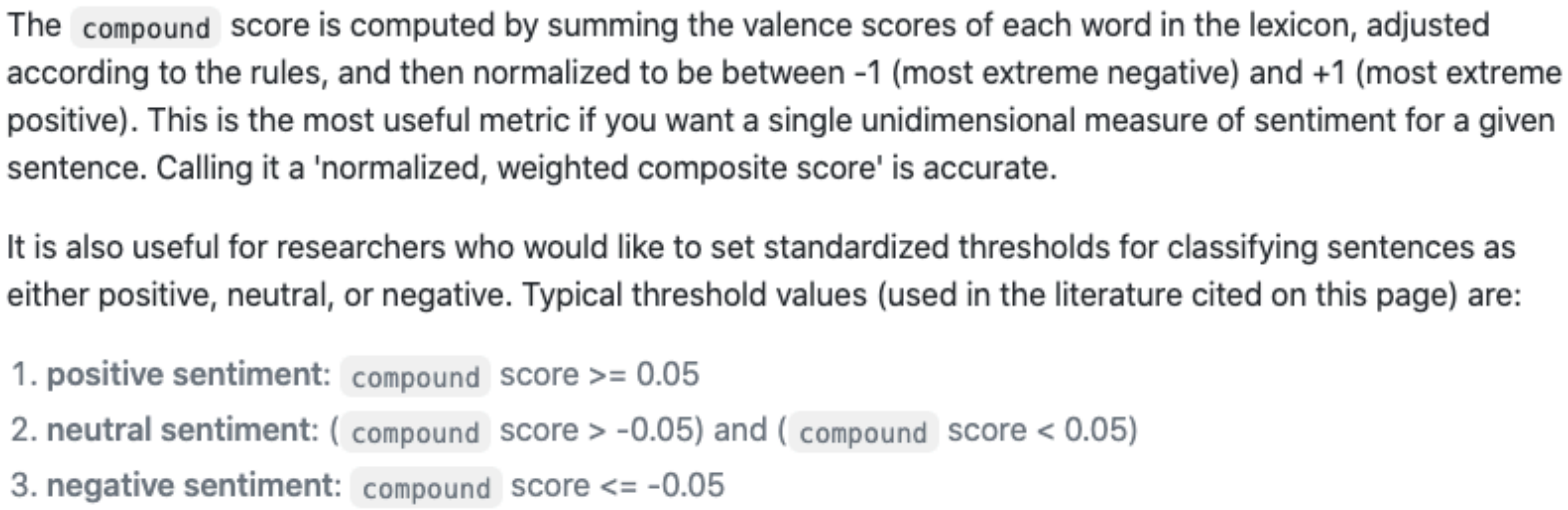
- Google’s BERT (Bidirectional Encoder Representations from Transformers, 2018) assign a vector to a word as a function of the entire sentence, therefore, the word can have different vectors based on different contexts.

- By far the most promising for assessing nuanced sentiment, but computationally very intensive. Still struggles with sarcasm.

Scoring Sentiment- Text Blob and VADER

- Lexicon-based sentiment analysis assigns scores to bags of words based on a pre-defined dictionary of positive and negative words. It will calculate sentiment scores over sentences.
- We would like to detect polarity (positive vs. negative opinion) in text. This very difficult when we consider that one sentence may contain multiple sentences and conditional statements. ("The acting was good, but the movie could have been better.")
- TextBlob will give us sentiment and intensity estimates. Here, we want opinion and polarity or intensity of the claim. VADER is thought to work better on Twitter data. It incorporates more emojis and common abbreviations/ slang. Some of these algorithms outperform human hand coding of sentiment.
- VADER expresses sentiment based on a compound score. It is within the NLTK library and is applied directly to unlabeled data. VADER relies on a dictionary to map lexical features to emotion intensities known as sentiment scores. It understands contextual negation "did not love" and emphasis through capitalization and punctuation (watch how you preprocess!)

Understanding Scoring

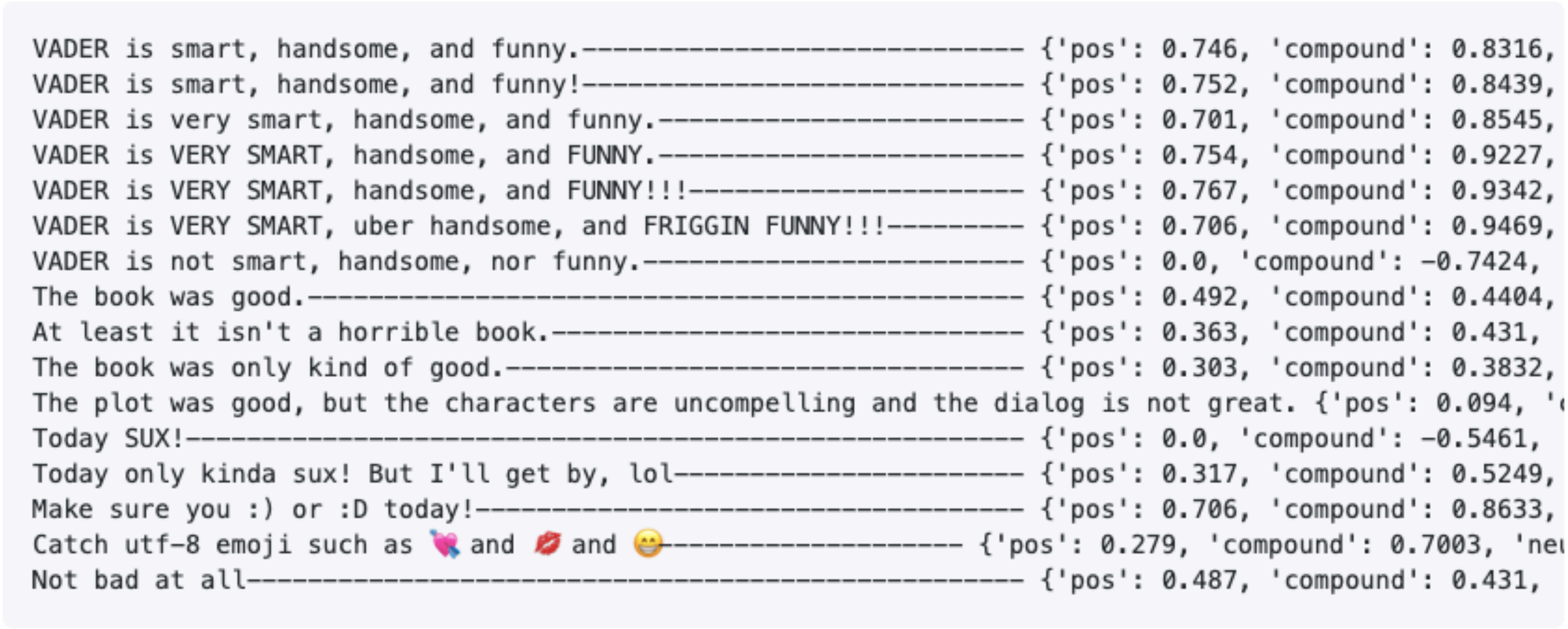


The `compound` score is computed by summing the valence scores of each word in the lexicon, adjusted according to the rules, and then normalized to be between -1 (most extreme negative) and +1 (most extreme positive). This is the most useful metric if you want a single unidimensional measure of sentiment for a given sentence. Calling it a 'normalized, weighted composite score' is accurate.

It is also useful for researchers who would like to set standardized thresholds for classifying sentences as either positive, neutral, or negative. Typical threshold values (used in the literature cited on this page) are:

1. **positive sentiment:** `compound` score >= 0.05
2. **neutral sentiment:** (`compound` score > -0.05) and (`compound` score < 0.05)
3. **negative sentiment:** `compound` score <= -0.05

Score Examples



```
VADER is smart, handsome, and funny.----------------------------------------------- {'pos': 0.746, 'compound': 0.8316,
VADER is smart, handsome, and funny!----------------------------------------------- {'pos': 0.752, 'compound': 0.8439,
VADER is very smart, handsome, and funny.------------------------------------------ {'pos': 0.701, 'compound': 0.8545,
VADER is VERY SMART, handsome, and FUNNY.------------------------------------------ {'pos': 0.754, 'compound': 0.9227,
VADER is VERY SMART, handsome, and FUNNY!!!---------------------------------------- {'pos': 0.767, 'compound': 0.9342,
VADER is VERY SMART, uber handsome, and FRIGGIN FUNNY!!!--------------------------- {'pos': 0.706, 'compound': 0.9469,
VADER is not smart, handsome, nor funny.------------------------------------------- {'pos': 0.0, 'compound': -0.7424,
The book was good.----------------------------------------------------------------- {'pos': 0.492, 'compound': 0.4404,
At least it isn't a horrible book.------------------------------------------------- {'pos': 0.363, 'compound': 0.431,
The book was only kind of good.---------------------------------------------------- {'pos': 0.303, 'compound': 0.3832,
The plot was good, but the characters are uncompelling and the dialog is not great. {'pos': 0.094, '
Today SUX!------------------------------------------------------------------------- {'pos': 0.0, 'compound': -0.5461,
Today only kinda sux! But I'll get by, lol----------------------------------------- {'pos': 0.317, 'compound': 0.5249,
Make sure you :) or :D today!------------------------------------------------------ {'pos': 0.706, 'compound': 0.8633,
Catch utf-8 emoji such as 💘 and 💋 and 😁-------------------- {'pos': 0.279, 'compound': 0.7003, 'ne
Not bad at all--------------------------------------------------------------------- {'pos': 0.487, 'compound': 0.431,
```

Concluding Thoughts

Lexical approaches like VADER can be good when:

- Domain-specific problem
- Lack of labeled data
- Limited ability to train on a large, relevant corpus

VADER and Textblob are optimized for social media. VADER works well with emojis.