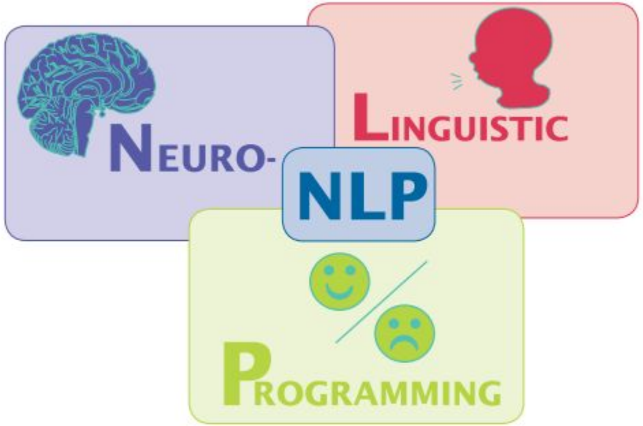
Bag of Words for Natural Language Processing

Intro to NLP



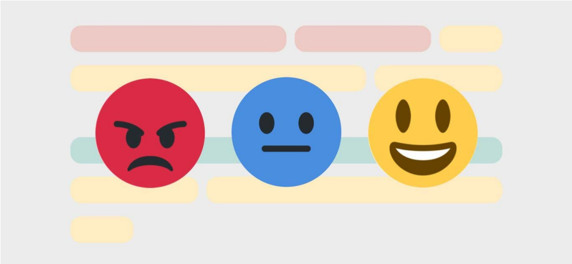
- Natural Language Processing (NLP) helps us draw insight from text.
- Computers are not able to easily process and analyze text like humans do.
- Reading a paragraph can easily make a thesis or main idea clear to a human reader, but a computer can do no such thing

Types of NLP Problems

Can we predict ratings based on text?

- Sentiment Analysis
    - Identifying subjective information and affective states in a piece of text
- Text Classification
    - Identifying with category or class a piece of text belongs to
- Document Summarization
    - Create a subset that represents the most important and relevant features of a document

Sentiment Analysis



- One of the goals of NLP is to draw meaning and sentimental analysis from text.
- Frequency of words can help a computer better understand sentiment of text
- Author’s sentiment from a text can help a computer learn what texts correspond to what sentiments.

Sentiment Analysis Example

Train Set of Review Data:

1. “I love using the new Google Pixel!”
2. “The pixel battery life is awfully short”
3. “Out of all my phones, the Pixel is average”

First, train model on a variety of different reviews that have different sentiments.

Test Set of Review Data:

1. “The pixel is easy to use. I love this phone.”
2. “I will never use this phone again!”
3. “The pixel is okay”

After training, the model should be able to accurately predict that 1 belongs to good sentiment, 2 belongs to bad sentiment, and 3 is neutral.

How to Create a Model for Sentiment Analysis?

Bag-of-Words Featurization

- The Bag of Words Model is commonly used in NLP to represent text in a document.
- The Bag of Words Model is a *featurization* that can be used with any learning model
- Map each unique word in the document with the number of times it occurs.
- Ordering or context of where words occur is ignored

Bag-of-Words Important Terms

- **Document**: one piece of data we are analyzing such as movie reviews, tweets etc. that will have associated labels

- **Corpus**: the collection of all available documents

- **Vocabulary**: the collection of all distinct words in the corpus

Bag-of-Words Process

- We first look over the corpus to develop the vocabulary which will be used to produce the feature vector that will be used for learning
- We then iterate over the training documents and create feature vectors based off the frequency of words in the document
- Words that are not in the vocabulary are not included in these features
- We then train and test the model (ex. SVM)

Bag-of-Words Example

1. For example, given the following document:

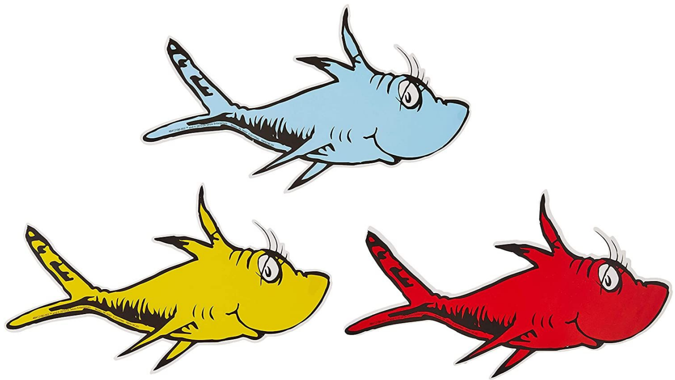
Professor Sahai tweets: *"one fish two fish red fish blue fish"*

2. Given a vocabulary of:

[the, dog, fish, red, blue, one, two]

3. The bag-of-words featurization would be:

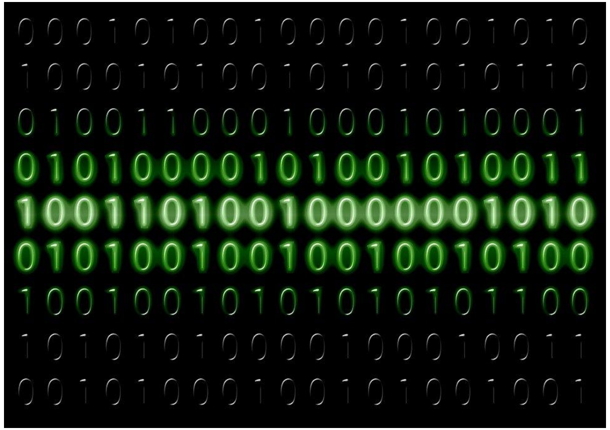
[0,0,4,1,1,1,1]



Optimizations

- There is room to optimize our featurization, especially given the situation
- Potential issues include:
  - We include too many words in our featurization, many of which do not add value
  - We are impacted by words that have various frequencies
  - We do not account for interactions between words that may change their meaning
- We now cover modifications and optimizations do deal with these

Binary Bag-of-Words



- There are many situations where we only want to know if a word was included in the document or not
- We do not care how many times it was stated, simply whether it was or not
- You will explore cases in which this may be optimal in the homework
- In order to do this instead of having a vector containing the count of each word, we only store a 1 if it is included and a 0 if it is not

Binary Bag-of-Words Example

1. For example, given the following document:

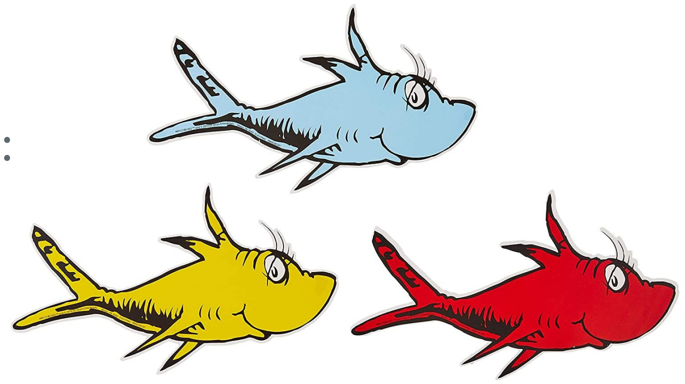
Professor Sahai tweets: *"one fish two fish red fish blue fish"*

2. Given a vocabulary of:

[the, dog, fish, red, blue, one, two]

3. The binary bag-of-words featurization would be:

[0,0,1,1,1,1,1]

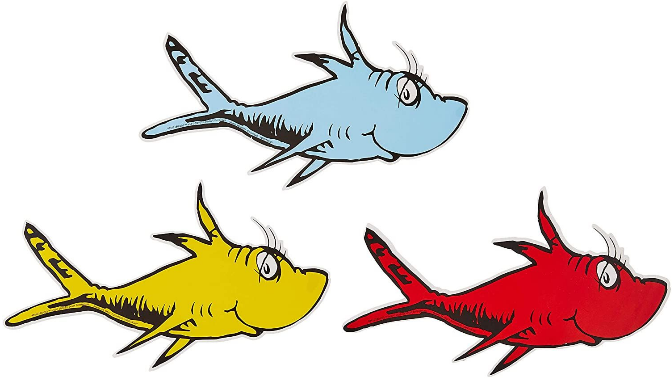


Feature Negation

- Many times we have phrases that negate meaning
  - “Not happy”, “Couldn’t wait”, “Wasn’t pleased”
- The classic bag-of-words model treats such words as just two separate words (ex. “Not” and “Happy”)
- To deal with this we treat such words as a feature of their own
- One way to do this is to double the size of the feature array and include the negations of all the words
- Whenever a word follows a “not” or a word ending in “‘nt” we negate the word

N-gram

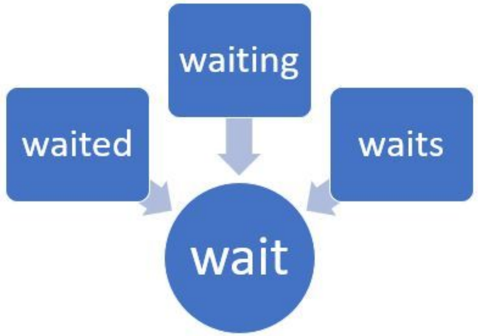
- N-gram: sequence of n words
- Similar to bag-of-words, but use n-grams instead of single words
- “One fish Two fish Red fish Blue fish” 2-gram model:
    - "One fish", "fish Two", "Two fish", "fish Red", "Red fish", "fish Blue", "Blue fish"
- N-grams provide insight to structure
    - can be better model than standard bag-of-words



Reducing Size of Vocabulary

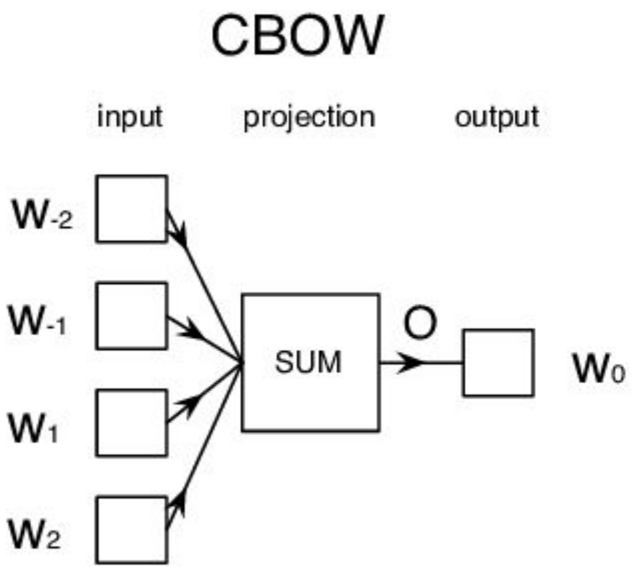
- Problem: bag-of-words vector can get very large, too large
- Ignore capitalization and discard punctuation
    - “Fish” == “fish” == “fish,”
- Help give a more accurate representation

Stemming and Stop Words



- Stemming is simply reducing all words to their stem
    - “process”, ”processes”, ”processed”, and ”processing” all have the same stem “process”
    - reduces vocabulary size, make the model better reflect the meaning of a document
- Remove all stop words to help reduce vocab
    - i.e. ”a”, ”the”
    - they don’t provide much meaning to the text, can be filtered out

Continuous Bag of Words



- A word embedding for a Word2Vec model
  - Continuous skip gram is another option
- Pick a window size to grab context words
- Adds more complexity to the model
  - relies more on contextual and semantic similarities and differences between words and texts
- Predict a target word based on the context words

CBOW Example

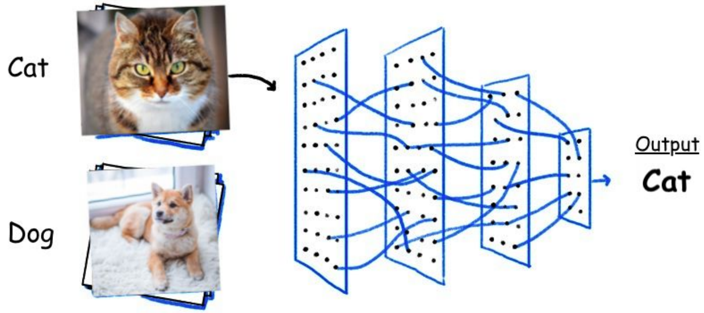
Sentence: "I will have green eggs and ham for breakfast, with orange juice on the side"

Window Size: 2
For each target word in the sentence, here are the corresponding context words:
Target : Context

I : [will, have] **will : [I, have, green]** **eggs : [have, green, and, ham]**
orange : [breakfast, with, juice, on]

Use these pairs to train a model to predict target words and use this to implement sentiment analysis

Other Use Cases

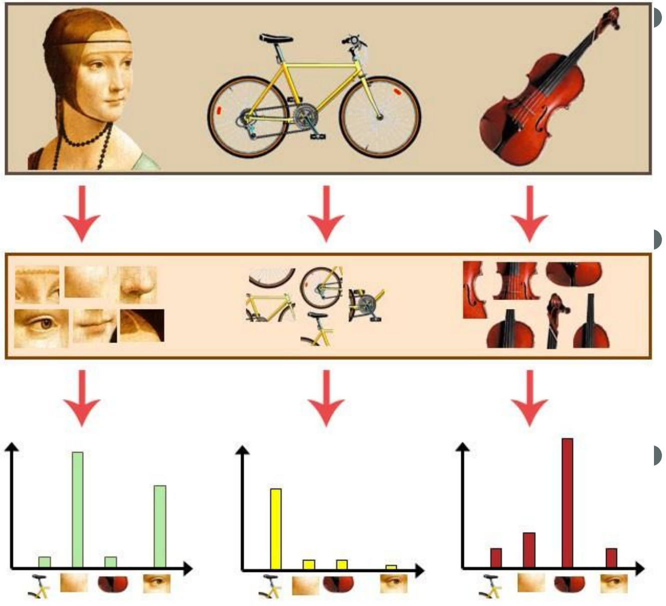


- Can use in image classification
- Bag-of-visual-words model
- Same concept as bag-of-words, different use case

Computer Vision

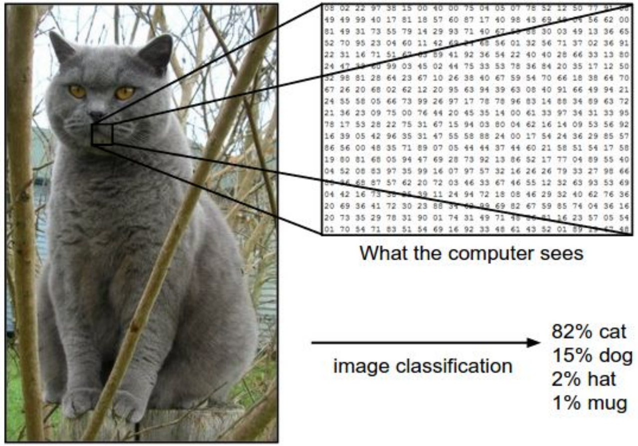
- Replace documents with images
- Replace words in documents with parts of an image
- We can then train on bag-of-visual-words features and classify images

Image Vocabulary



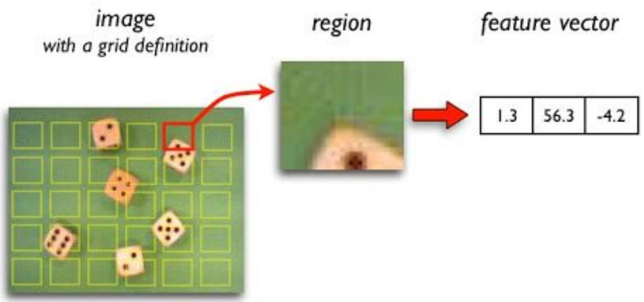
- Parts of the image make up its vocabulary
  - violin has very unique curvature
  - human face positioning of the nose, eyes and mouth, etc.
- Piece of text is defined by all of its individual words
- Image is defined by its defining structure and composition

Feature Extraction



- Regular Grid
- Interest Point Detection
- Random Sampling

Regular Grid



- Image is converted into a structured grid
- Each feature of the image is one of these rectangular parts
- Vocabulary is the sum of all these parts of the grid

Interest Point Detection

- Splitting up an image in terms of its most stable and formed parts
  - different points of interest = different features
- Corners and textures can be used as points of interest
  - likely to be used to categorize an image

Random Sampling

- Randomly sampling features from an image
  - Can achieve equal or greater image classification accuracy compared to deliberate models
- Often more discriminatory between features and images

tf-idf

- Problem: most frequently occurring words tend to dominate bag-of-words
- *Term Frequency-Inverse Document Frequency* (tf-idf)
  - tf-idf applies a weight to each word based on the term and document frequency
- **Term frequency**: frequency of a word within a document
- **Document frequency:** frequency that a word occurs at least among all the documents.

tf-idf Cont.

- If analyzing sports news articles, the word points likely occurs in most of the articles, so it's not very useful in distinguishing between two different articles

For word *w* in document *d*, one common weighting scheme is:

$$(1 + \log tf) \cdot \log(1 / (df / N))$$

Where:

Tf: # of words in *d*, ***Df***: # of documents which *w* occurs in

N: # of documents we are analyzing