**NLP**

Language models can be categorized into several types:

Statistical Language Models: Based on probability distributions, often using n-grams to predict the likelihood of a word given its context.

Neural Language Models: Utilize neural networks, such as recurrent neural networks (RNNs) or transformer models, to capture complex patterns and dependencies in text data.

Rule-based Language Models: Employ explicit rules or grammatical structures to generate or interpret language, often used in traditional parsing techniques.

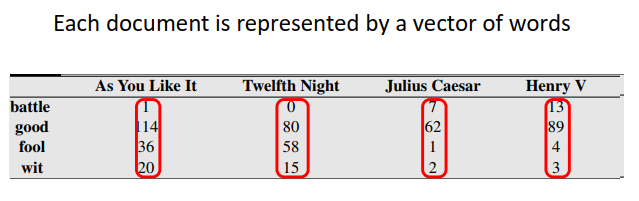
Hybrid Language Models: Combine elements of statistical, neural, and rule-based approaches to leverage their respective strengths in language understanding and generation tasks.

Stemming and lemmatization are both text processing techniques used to reduce words to their base forms. Although stemming is faster but least accurate so **Lemmatization is preferred**.

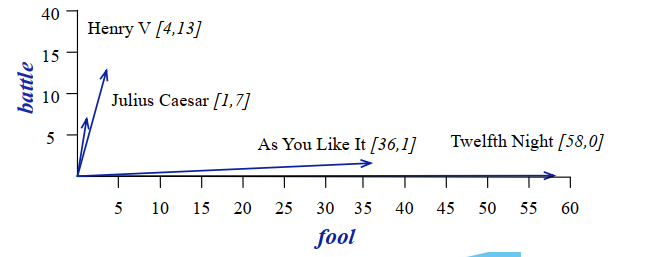
**Text Classification with Scikit-Learn:**

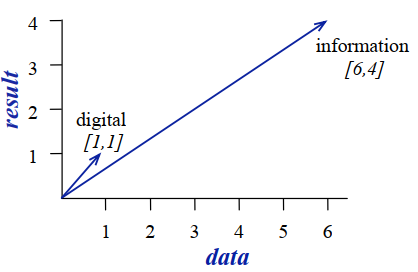
We cannot work with text directly when using machine learning algorithms. Instead, we need to convert the text to numbers. We call vectorization the general process of turning a collection of text documents into numerical feature vectors. This specific strategy (tokenization, counting and normalization) is called the Bag of Words or “Bag of n-grams” representation. We define a word as a vector called an "embedding"

Term Vector Embedding



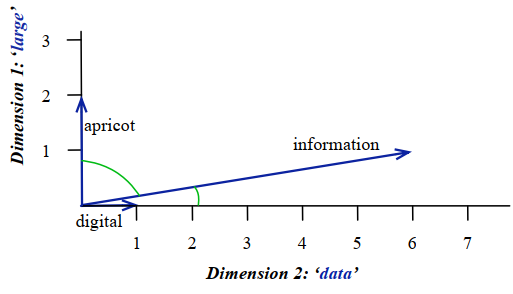
In the above pic, these words have these many occurences in these 4 movies (column names). Visualizing these docu vectors results in:

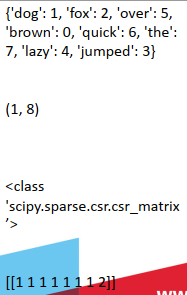


This shows the tendency of belonging to a funny movie or war one (x and y axis). Two words are similar in meaning if their context vectors are similar as in pic:

Cosine for Computing Similarity:

a ·b *=* |a||b| *cos* *Ɵ*

Where Ɵ is the difference.

**Converting Text to Numbers**

from sklearn.feature\_extraction.text import CountVectorizer

# list of text documents

text = ["The quick brown fox jumped over the lazy dog."]

# create the transform

vectorizer = CountVectorizer()

# tokenize and build vocab

vectorizer.fit(text)

# summarize

print(vectorizer.vocabulary\_)

# encode document

vector = vectorizer.transform(text)

# summarize encoded vector

print(vector.shape)

print(type(vector))

print(vector.toarray())

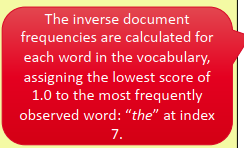
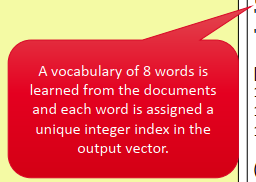
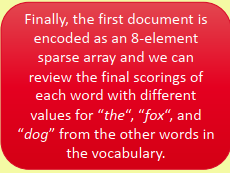
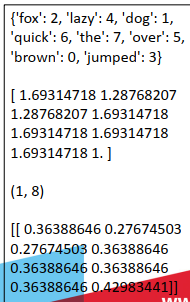
Convert vector to array:

vector = vectorizer.transform(text2)

print(vector.toarray())

Tf-idf (**Term Frequency-Inverse Document Frequency**) aims to quantify the importance of a word in a document relative to the entire document collection. **Term Frequency: This summarizes how often a given word appears within a document. Inverse Document Frequency: This downscales words that appear a lot across documents.** The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings. if you already have a learned CountVectorizer, you can use it with a TfidfTransformer to just calculate the inverse document frequencies and start encoding documents

**tf-idf value for word t in document d: wt,d = tft,d X idft**

**Word Counts with CountVectorizer**

from sklearn.feature\_extraction.text import TfidfVectorizer

# list of text documents

text = ["The quick brown fox jumped over the lazy dog.",

"The dog.",

"The fox"]

# create the transform

vectorizer = TfidfVectorizer()

# tokenize and build vocab

vectorizer.fit(text)

# summarize

print(vectorizer.vocabulary\_)

print(vectorizer.idf\_)

# encode document

vector = vectorizer.transform([text[0]])

# summarize encoded vector

print(vector.shape)

print(vector.toarray())

The scores are normalized to values between 0 and 1 and the encoded document vectors

can then be used directly with

most machine learning

algorithms.

**Statistical Lang Modeling**

Enumeration: the action of mentioning a number of things one by one.

"the complete enumeration of all possible genetic states"

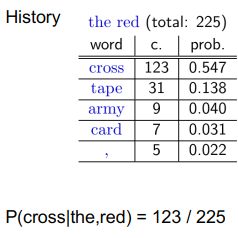
The probability of a sentence is multiplication of all the probabilities of words in it, known as **unigram language model. But the issue with this model is that it does not maint a seq as req in a sentence.**

**Combining Language Models (e.g. uni and bigram)**.

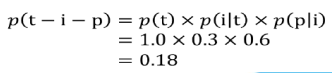
We combine various language models to improve the accuracy and robustness of predictions by leveraging different aspects of language structure and context.

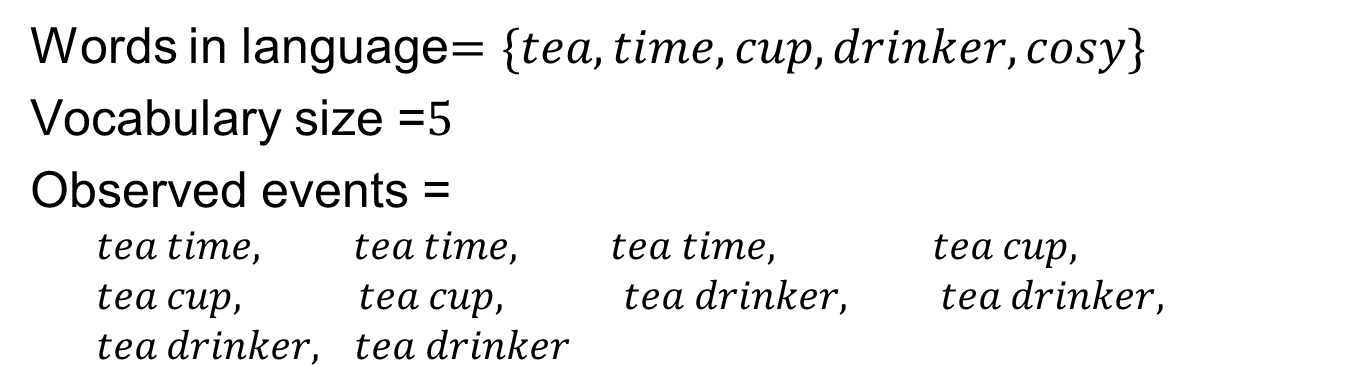
**Probability of an n-gram**

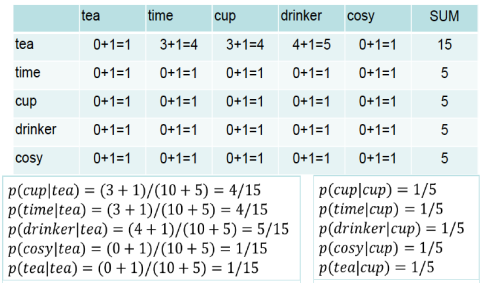
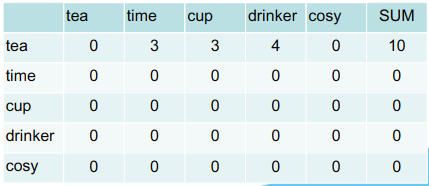
So more no of words are used.e.g. 2-gram …n-gram..along with their probabilities. E.g. 3-gram model involving 3 words is below. Where the occurrence of red was calculated 225. So:-



Similarly, for a bigram of characters tip :-



**But the issue in this approach is words with zero probability wrt other words. This can be handled by using:- 1- Count adjustment, 2- Interpolation, 3- Back-off, 4-One-Smoothing (adding +1 to everything):-**



**If earlier the probability of tea and cosy was 0 i.e. p(cozy|tea) = 0/10 = 0, now it will be 1/15. So,**

**p(tea cozy) will be = p(tea) x p(cozy|tea) = 10/20 x 15/30 while p(tea) has been calculated using unigram.**

Higher and lower order n-gram models have different strengths and weaknesses:-

- high-order n-grams

* sensitive to more context
* sparse counts

- low-order n-grams

* very limited context
* have robust counts

**Find smoothing in Lab3 LSM**

**Interpolation**: Let's say we have two language models, one based on unigram probabilities and another based on bigram probabilities. We can interpolate these models to get a smoothed probability for given seq of words. Suppose our unigram model has probabilities: P(tea) = 0.4, P(cozy) = 0.3. And bigram model has probabilities: P(tea|cozy) = 0.6, P(cozy|tea) = 0.7. We can interpolate these probabilities using a weighted average. Let's assume we use equal weights for both models (0.5 each): Interpolated probability for "tea":- Interpolated p(tea) = 0.5 x unigram p(tea) + 0.5 x bigram p(tea) = 0.5x0.4+0.5x 0.7 = 0.55, similarly, for p(cozy) = 0.45. **Interpolation helps to mitigate the sparsity problem in language modeling by combining the strengths of different n-gram models.**

**Note: A language corpus is a large and structured collection of text or speech data, used for linguistic analysis, natural language processing tasks, and machine learning. Structure incls: metadata such as the source of the text, the date it was written or recorded, and any linguistic annotations such as part-of-speech tags or syntactic parses. Parse means breaking down a sentence to determine its syntactic components, such as nouns, verbs, adjectives, and the relationships between them.**

Wk 4: Lambda is either zero or 1 in lam P(w). N is number of words. In this context of filling the blanks, what is the best approach? Singleton: used only one time. **N Grams is N-1 number of words**.

**Back off smoothing**. Trust the higher n-gram model in combined multiple models.

**Neural LMs**

**Continuous Space LMs are also known as Neural Netwrok LMs. Neural Network trained to predict a new**

**word (vector) given a sequence of words (vectors).**

**What is Feature Extraction/Vectorization?** Words need to be encoded as integers or floating point values for use as input to a machine learning algorithms, this process is called feature extraction or vectorization.

How many **n\_features** in the bags of words representation? The bags of words representation implies that n\_features is the **number of distinct words in the corpus**. Raw frequency or occurrences of words too many times is bad representation.e.g. it, the etc. So we use tf and idf.

Short vectors may be easier to use as features in machine learning (less weights to tune)

• Dense vectors may generalize better than storing explicit counts

e.g car and automobile are linked words this can be worked out better by dense vectors

Word2vec, Fasttext, Glove are a few examples of dense embeddings.

**LLMs**

LLM (Large Language Model) is a type of language model that utilizes large-scale neural networks to generate and understand natural language text, exhibit stronger language understanding and generation abilities, enabling tasks such as text completion, translation, summarization, and more.

These powerful language models capable of understanding and generating human-like text. They typically contain millions or even billions of parameters and are trained on vast amounts of text data.

**Types**:

**Autoregressive Models**: Autoregressive models generate text one token at a time, conditioning each token on previously generated tokens. Examples include GPT (Generative Pre-trained Transformer) models, which predict the next word in a sequence given all previous words.

**Autoencoding Models**: Reconstruct input text, encouraging the model to learn a compact representation; examples include BERT (Bidirectional Encoder Representations from Transformers).

**Transformer Models**: Utilize self-attention mechanisms to capture dependencies across input tokens effectively; examples include GPT and BERT architectures.

**By design**, there are few other types; encoder only, decoder only, encoder-decoder.

**Vectorization**: is the representation of words into vectors using methods such as one-hot encoding, word embedding, contextual embedding.

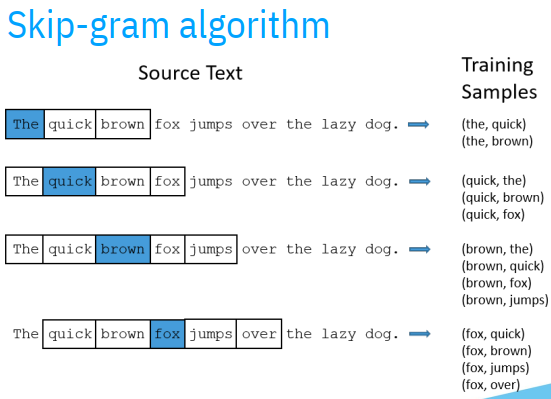
e.g. cat is represented as a dense vector [0.2, 0.4, -0.1, ...], capturing its semantic meaning in a continuous vector space.

**One-Hot Encoding**: Representing each word as a binary vector where only one element is 1 (indicating the word's position in the vocabulary).

**Word Embeddings**: Dense, real-valued vectors learned from large text corpora using models like **Word2Vec**, GloVe, or FastText, where similar words have similar vector representations.

**Contextual Embeddings**: Embeddings that capture context-dependent meanings of words, such as ELMo, GPT, and BERT, which consider the surrounding context when generating word vectors.

Word2Vec represents words as dense vectors in a continuous vector space, trained using neural networks to capture semantic relationships between words based on their co-occurrence patterns in a large corpus of text data. Continuous Bag of Words (CBOW) and Skip-gram are used by W2V. While CBOW predicts a target word based on its context words, **Skip-gram predicts context words given a target word**.

****

1. Treat the target word and a neighboring context word as positive examples.

2. Randomly sample other words in the lexicon to get negative samples

3. Use logistic regression to train a classifier to distinguish those two cases

4. Use the weights as the embedding z = w · x+b

Sigmoid function can be used to get output of this neural nw between 0 and 1.

**Exp**:

Corpus: "I love green tea."

Vocabulary: {"I", "love", "green", "tea", "."}

Training pairs (target word, context word):

("I", "love"), ("love", "I"), ("love", "green"), ("green", "love"), ("green", "tea"), ("tea", "green")

**Use of existing word embedding is better. E.g. GoogleNews vectors.**

**Link 2 notebook for code**

**The positive parameter contains a list of words whose vectors should be added together.**

**The negative parameter contains a list of words whose vectors should be subtracted from the sum of vectors specified in positive.**

**Embeddings can help in studying history because it shows past and present association of words.**

**Embeddings = vector models of meaning**

**Sentiment Analysis**

Emotions are taken as either atomic units or dimensions. Various Sentiment Lexicons are available to study emotions defining emos in various ways. (slides). Emo factors are:

* valence (the pleasantness of the stimulus)

• arousal (the intensity of emotion provoked by the stimulus)

• dominance (the degree of control exerted by the stimulus)

**A lexicon refers to a collection of words or phrases along with their associated attributes, such as part-of-speech (POS) tags, semantic categories, sentiment scores, or other linguistic properties.**

Lexicons are often used as resources for various NLP tasks, including text analysis, information retrieval, sentiment analysis, and machine translation. They provide a structured representation of linguistic knowledge that can aid in understanding and processing natural language text.

WordNet: an online thesaurus could be used to find polarity of words (+ -).

**semi-supervised Lexicon Learning**

Models are trained using clusters made from words wrt polarities:-

* Using “and” and “but”

• Using words that occur nearby in the same document

• Using WordNet synonyms and antonyms

Supervised Learning can also be used by Review Datasets such as Trip Advisor, IMDB etc. Each review has a score (e.g. 1-5 stars or 1-10). Just need to count word freq with each score and then normalize.

* Train a classifier based on supervised data
  + Predict: human-labeled connotation of a document
  + From: all the words and bigrams in it
* Use the regression coefficients as the weights

**Lexicons for detecting document affect:**

**Simplest supervised method**

• Build a classifier

* + Predict sentiment (or emotion, or personality) given features
  + Count lexicon categories” and use them as a features
  + Sample features:
  + LIWC lexicon category “cognition” had count of 7
  + NRC Emotion category “anticipation” had count of 2

• Baseline

* + Count all words and bigrams in the training set then ML used as a baseline for comparison with more complex models.
  + This is hard to beat
  + But only works if the training and test sets are very similar

Personality Detection could be done using these.

LIWC is Linguistic Inquiry and Word Count software used for text analysis.

**Summary: Connotation in the lexicon**

• Words have various connotational aspects

• Methods for building connotation lexicons

Based on theoretical models of emotion, sentiment

• By hand (mainly using crowdsourcing)

• Semi-supervised learning from seed words

• Fully supervised (when you can find a convenient signal in the world)

• Applying lexicons to detect affect and sentiment

• **Unsupervised**: pick simple majority sentiment (positive/negative words)

• **Supervised**: learn weights for each lexical category

**Differences between Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs):**

|  |  |  |
| --- | --- | --- |
| Feature | Recurrent Neural Networks (RNNs) | Convolutional Neural Networks (CNNs) |
| Data Type | Designed for sequential data (e.g., text, time series) | Primarily used for grid-like data (e.g., images), but can be adapted for sequential data |
| Architecture | Contains loops to process sequential data recursively | Consists of convolutional layers for extracting spatial patterns |
| Temporal Dynamics | Capable of capturing temporal dependencies across time steps | Efficient at capturing local patterns, but less effective at modeling long-range dependencies |
| Long-term Dependencies | May suffer from the vanishing **gradient problem over long sequences** | Typically not designed to capture long-range dependencies effectively |
| Translation Invariance | Not inherently suited for capturing translational invariance | Highly effective at capturing translational invariance |
| Applications | Natural language processing (e.g., language modeling, translation) | Image processing (e.g., classification, object detection), text classification (with modifications) |
| Example Use Cases | Speech recognition, machine translation, sentiment analysis | Image classification, object detection, text classification (with modifications) |