

Introduction to Modern AI

Week 7: Natural Language Processing

Gavin Hartnett

PRGS, Winter Quarter 2022

Overview

- 1 Working with Text Data
- 2 Naive Bayes and the Bag of Words Assumption
- 3 TF-IDF
- 4 Vector Embeddings and Representation Learning
- 5 Modern NLP

Motivation

- There is a wide range of useful applications for computer systems that can process and to some extent understand natural language
 - search
 - translation
 - spam filters
 - text completion suggestions
 - AI assistants and chatbots
 - automatic captioning, transcription
 - information extraction
 - and so on
- Natural language is fundamental to how we (humans) think, reason, and communicate with one another
- Therefore, NLP could be an important area for developing “true” artificial intelligence

Working with Text Data

Numerical Representation of Textual Data

- How should we represent textual data?

The quick brown fox jumps over the lazy dog

- Word level:

[The, quick, brown, fox, jumps, over, the, lazy, dog]

- Character level:

[t, h, e, , q, u, i, c, k, , b, r, ...]

- Quick observations:

- The data is a sequence of symbols. Order matters!
- There are upper, lower-case letters, punctuation, special characters, etc

ASCII

- The precise way we will numerically represent the data will depend on our application, choice of methodology
- Typically this representation differs from how the computer stores the data
- The American Standard Code for Information Interchange (ASCII) developed for telegraph communication
- Assigns a number 0-127 ($7 \text{ bit} = 2^7 = 128 \text{ options}$) to each character
- Tailored to Latin alphabet, English in particular
- Includes some non-printing control codes that are now obsolete

ASCII Table

ASCII TABLE

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---------|-----|------------------------|---------|-----|---------|---------|-----|------|---------|-----|-------|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | \$ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | (| 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 |) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [| 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D |] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

Image source: <https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg>

Unicode

- ASCII falls short in many ways
 - What about other languages (e.g., Mandarin)?
 - What about Emojis?
 - Unicode is another approach
 - Currently captures 144,697 characters, easy to add new ones
- Unicode has multiple implementations (UTF-8, UTF-16, etc)
- UTF-8 is most dominant worldwide
- For the most part in doing NLP you won't need to worry about ASCII, Unicode, but occasionally it will be important
- Mainly becomes an issue during the initial preprocessing/downloading of textual data

Vectorization and N -Gram Representations

- Goal: convert text sequence into a *sparse* numerical vector
- sparse means that most entries will be zero
- General Strategy:
 - Introduce vocabulary \mathcal{V} containing $N = |\mathcal{V}|$ distinct tokens
 - 1-gram: vocabulary consists of distinct words
 - 2-gram: vocabulary consists of distinct pairs of words
 - N -gram: vocabulary consists of distinct sequences of N words

Example: 1-Gram

$V = \{\text{the, quick, brown, fox, jumps, over, lazy, dog}\}$

| word | vector | | | | | | | |
|-------|--------|-------|-------|-----|-------|------|------|-----|
| | The | quick | brown | fox | jumps | over | lazy | dog |
| the | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| quick | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| brown | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| fox | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| jumps | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| over | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| The | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| lazy | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| dog | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Example: 2-Gram

- *The quick brown fox jumps over the lazy dog*
- Vocabulary is:

$$\mathcal{V}_2 = \{\text{the quick, the brown, ..., quick brown, quick fox, ..., lazy dog}\}$$

- If there are n words, 2-gram vocab has size $N = n(n - 1)/2$
- Retains a bit more of the structure than 1-gram (e.g., 'lazy dog' is a distinct term, not just 'lazy' and 'dog' separately)
- Vocab size grows exponentially with number of graphs, quickly becomes impractical

$$N_p = |\mathcal{V}_p| = \binom{N}{p}$$

- Typically incorporate lower-grams, i.e. use 1-gram plus 2-gram
- higher-grams might be only partially added (i.e., include a few common 3-grams)

Tokenization

- Tokenization is the process of breaking up text into constituent parts (tokens)
- We've already seen an example of this:
The quick brown fox jumps over the lazy dog
[The, quick, brown, fox, jumps, over, the, lazy, dog]
- There are multiple ways to perform the tokenization, and the way this is done can have important implications
- For example: how should punctuation be handled? What about white-spaces?
- Modern NLP approaches actually learn a tokenizer

Stemming and Lemmatization

- Stemming and Lemmatization are similar, and can help improve performance by reducing the number of distinct words in the vocabulary
- E.g., sit, sitting, sat are all distinct words but they share a common meaning called the *lexeme*
- One word is chosen as the *lemma*, or representative of the lexeme
- Stemming and Lemmatization both reduce words down to lemma form
- Stemming acts only on individual words (1-grams), and is quick
- Lemmatization can handle words whose meaning is ambiguous without context, such as 'meeting' (noun or verb?)

Other Preprocessing Issues

- Punctuation should be handled. One idea is to convert all to lower-case.
- Special characters, url links, email address, hashtags, etc must be handled.
- Stop words (e.g., is, the, like, ...) might be removed
- Useful Python libraries exist for these preprocessing steps:
 - [Natural Language ToolKit \(NLTK\)](#)
 - [HuggingFace](#)

Naive Bayes and the Bag of Words Assumption

Text Classification: Naive Bayes

- Let's now consider how we might go about classifying text documents
- Each data point/observation is a document
- document consists of tokens (words)
- We'll represent this as a series words/tokens:

$$d = \{w_1, w_2, \dots, w_n\}$$

- We want to model $P(c|d)$
- Model $P(d|c)$ directly and use Bayes theorem

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

- Example of generative (vs discriminative) approach

Text Classification: Naive Bayes

- Goal: model for $P(c|d)$
- Use Bayes theorem and model $P(d|c)$ directly:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

- Iteratively apply $P(x, y|c) = P(x|y, c)P(y|c)$ rule:

$$\begin{aligned}P(d|c) &= P(w_1, w_2, w_3, \dots, w_n|c) \\&= P(w_1|w_2, w_3, \dots, w_n, c)P(w_2, w_3, \dots, w_n|c) \\&= P(w_1|w_2, w_3, \dots, w_n, c)P(w_2|w_3, \dots, w_n, c)P(w_3, \dots, w_n|c) \\&= \dots\end{aligned}$$

- assume conditional independence (bag of words):

$$P(w_i|w_{i+1}, \dots, w_n, c) = P(w_i|c)$$

- Then

$$P(d|c) = \prod_{i=1}^n P(w_i|c)$$

Text Classification: Naive Bayes

- Main challenge of NLP is how to capture/handle the temporal correlations in the sequence
- Naive Bayes/Bag of Words is the simplest way to represent text once it has been tokenized - ignore the time ordering completely

$$P(d|c) = \prod_{i=1}^n P(w_i|c)$$

- How to model/estimate $P(w_i|c)$?

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c)}{\sum_{w \in \mathcal{V}} \text{count}(w, c)}$$

- Here $\text{count}(w, c)$ just counts the number of times word w appears in documents of class c

TF-IDF

Information Retrieval and TF-IDF

- Naive Bayes utilizes raw counts/frequencies
- For many applications, raw counts/frequencies are too naive
- Information Retrieval
 - Example: return all documents most relevant to the phrase “machine learning is hard”
 - “is” is a very common word, many documents will have high occurrences
- TF: term frequency

$$\text{tf}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

- IDF: inverse document frequency

$$\text{idf}(t, \mathcal{D}) = \log \left(\frac{|\mathcal{D}|}{|d \in \mathcal{D} : t \in d|} \right)$$

- TF-IDF

$$\text{tf-idf}(t, d, \mathcal{D}) = \text{tf}(t, d) \times \text{idf}(t, \mathcal{D})$$

Information Retrieval and TF-IDF

- TF-IDF

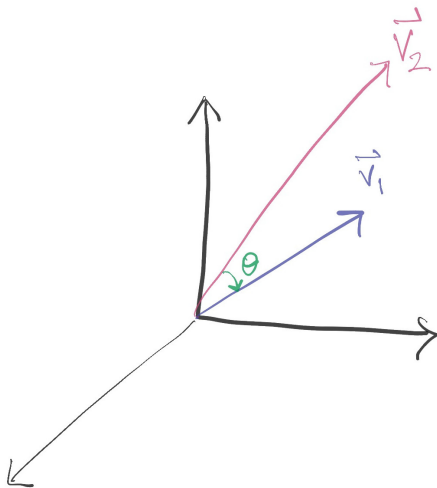
$$\text{tf-idf}(t, d, \mathcal{D}) = \text{tf}(t, d) \times \text{idf}(t, \mathcal{D})$$

- TF: how frequent is a term in a given document
- IDF: small for terms that show up in many documents
- TF-IDF values are often used as useful features to work with
- Can consider TF-IDF to be a dense vector feature for each *document*:

$$\mathbf{v}(d) = \{\text{idf}(t_1, \mathcal{D}), \text{idf}(t_2, \mathcal{D}), \dots, \text{idf}(t_n, \mathcal{D})\}$$

- Can now introduce a concept of distance/similarity between documents

Cosine Similarity



$$S_c(v_1, v_2) = \cos \theta$$

$$= \frac{v_1 \cdot v_2}{\|v_1\| \|v_2\|}$$

$S_c = 1$: parallel (similar)

$= 0$: orthogonal (unrelated)

$= -1$: anti-parallel (opposite)

Vector Embeddings and Representation Learning

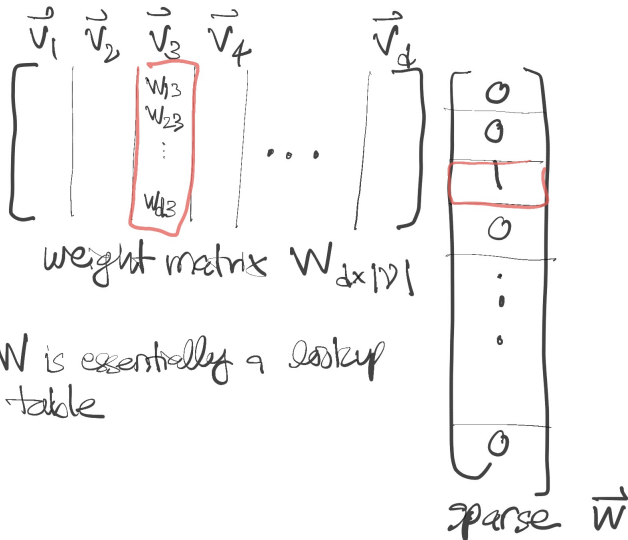
Vector Embeddings and Representation Learning

- The TF-IDF vectors are dense, compared to the sparse vectors we encountered before
- E.g., *The quick brown fox jumps over the lazy dog*
word-level: $v(\text{quick}) = \{0, 1, 0, 0, 0, 0, 0, 0, \dots\}$
Bag of Words doc-level: $v(d) = \{1, 2, 1, 1, 1, 1, 1, 1, \dots\}$
- The TF-IDF vectors contain less info than original data which makes them easier to work with
- Can act as useful features for down-stream tasks such as classification
- Big idea: *learn* the dense vector representations (embeddings)
- These vectors will not be useful in and of themselves, they will be useful for downstream tasks
- General approach is called *representation learning* or *feature learning*

Word2Vec

- Word2Vec is a very well-known approach for learning useful vector embeddings at the word level
- Most popular implementation is in the [Gensim](#) library
- The basic idea is the important first step in more sophisticated and modern language models

Word2Vec



W is essentially a lookup table

Word2Vec

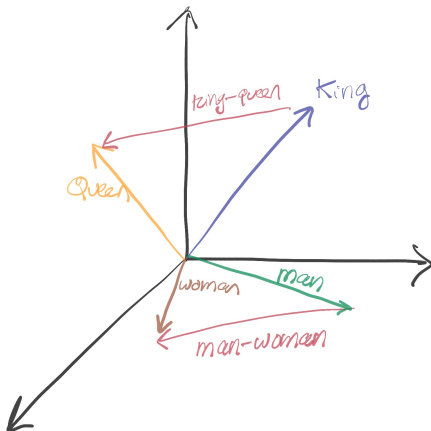
- There are two versions of Word2Vec (Skip-gram, Continuous Bag of Words)
- Embeddings are trained differently in the two versions, but both are based on utilizing *context*
- Both effectively use a 2-layer NN
- Both learn vector representations that are useful for predicting other words in a sentence
- If you are curious, here are some useful references for further reading:
 - [Original paper 1](#)
 - [Original paper 2](#)
 - [Lillian Weng's blog](#)
- There is also a document-level version called doc2vec

Word2Vec

- The learned vector embeddings can be shown to encode some interesting relationships
- Having mapped words to vectors, we can now perform algebraic operations on words!
- E.g., can subtract word vectors:

$$\mathbf{v}' = \mathbf{v}(\text{king}) - \mathbf{v}(\text{queen}) + \mathbf{v}(\text{woman})$$

- Can then search through vocabulary to find closest word vector to \mathbf{v}'
- Often the result makes sense: in this case it will be $\mathbf{v}(\text{man})$



Modern NLP

Deep Learning for NLP

- In this class we are just giving a short intro to NLP
- “Deep NLP” will be covered in more detail in the advanced class
- These next few slides will just set the stage and give a very general overview of the field

Neural Architectures for NLP

- There are several neural network architectures tailored for processing sequence/text data
- Recurrent Neural Networks (RNNs) can process sequences of arbitrary length
- In principle, they can capture long-term dependencies, but in practice this is difficult
- There are many variants aimed to address this and improve performance
- Basic “Vanilla” RNN: for $t = 0, 1, \dots$, compute

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}$$

- seq2seq: maps input $\{\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \dots\}$ to output $\{\mathbf{o}^{(0)}, \mathbf{o}^{(1)}, \dots\}$

Neural Architectures for NLP

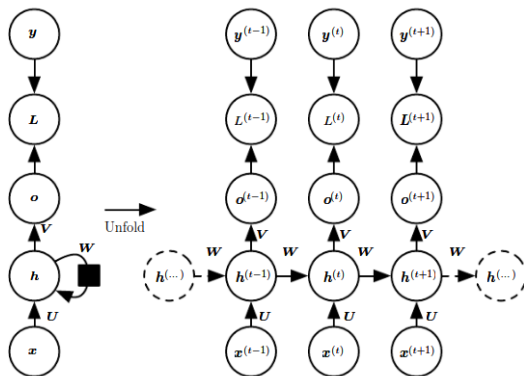


Figure 10.3: The computational graph to compute the training loss of a recurrent network that maps an input sequence of x values to a corresponding sequence of output o values. A loss L measures how far each o is from the corresponding training target y . When using softmax outputs, we assume o is the unnormalized log probabilities. The loss L internally computes $\hat{y} = \text{softmax}(o)$ and compares this to the target y . The RNN has input to hidden connections parametrized by a weight matrix U , hidden-to-hidden recurrent connections parametrized by a weight matrix W , and hidden-to-output connections parametrized by a weight matrix V . Equation 10.8 defines forward propagation in this model. (Left) The RNN and its loss drawn with recurrent connections. (Right) The same seen as a time-unfolded computational graph, where each node is now associated with one particular time instance.

Attention Is All You Need

- RNNs and variants have fallen out of favor relative to so-called transformer models
- Transformer models employ an operation called *attention* which allows the model to learn which parts of the sequence to focus on (context)
- This allows them to be more efficient compared to RNNs which process data in serial
- This in turns allows transformers to scale up to incredibly large model sizes
- Results in super large, expensive models called by many names
 - Sesame Street Models
 - Large Language Models
 - Foundation Models