The Problem with Text

- A problem with modeling text is that it is messy, and techniques like machine learning algorithms prefer well defined fixed-length inputs and outputs.
- Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers.
- In language processing, the vectors x are derived from textual data, in order to reflect various linguistic properties of the text. This is called **feature extraction or feature encoding.**

Feature encoding

- Bag-of-words (BoW)
- **TF-IDF**

bag-of-words model (BoW)

- A bag-of-words model, or BoW for short, is a way of extracting features from text for use in modeling, such as with machine learning algorithms.
- The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents.
- A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:
- A vocabulary of known words.
- A measure of the presence of known words.
- It is called a "*bag*" of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

- The intuition is that documents are similar if they have similar content. Further, that from the content alone we can learn something about the meaning of the document.
- The bag-of-words can be as simple or complex as you like. The complexity comes both in deciding how to **design the vocabulary of known words** (or tokens) and how to **score the presence of known words**.

Step 1: Collect Data

Below is a snippet of the first few lines of text from the book "A Tale of Two Cities" by Charles Dickens, taken from Project Gutenberg.

- It was the best of times,
- it was the worst of times,
- it was the age of wisdom,
- it was the age of foolishness,

For this small example, let's treat each line as a separate "document" and the 4 lines as our entire corpus of documents.

Step 2: Design the Vocabulary

- Now we can make a list of all of the words in our model vocabulary.
- The unique words here (ignoring case and punctuation) are:
  1. “it”
  2. “was”
  3. “the”
  4. “best”
  5. “of”
  6. “times”
  7. “worst”
  8. “age”
  9. “wisdom”
  10. “foolishness”
- That is a vocabulary of 10 words from a corpus containing 24 words.

Step 3: Create Document Vectors

- The objective is to turn each document of free text into a vector that we can use as input or output for a machine learning model.
- Because we know the vocabulary has 10 words, we can use a fixed-length document representation of 10, with one position in the vector to score each word.
- The simplest scoring method is to mark the presence of words as a boolean value, 0 for absent, 1 for present.
- Assuming we have a dictionary mapping words to a unique integer id, a bag-of-words featurization of a sentence could look like this:
  - It was the best of times = [1,1,1,1,1,1,0,0,0,0]
  - "it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
  - "it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
  - "it was the age of foolishness" = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]

- All ordering of the words is nominally discarded and we have a consistent way of extracting features from any document in our corpus, ready for use in modeling.

Managing Vocabulary

- You can imagine that for a very large corpus, such as thousands of books, that the length of the vector might be thousands or millions of positions. Further, each document may contain very few of the known words in the vocabulary.
- There are simple text cleaning techniques that can be used as a first step, such as:
  - Ignoring case
  - Ignoring punctuation
  - Ignoring frequent words that don't contain much information, called stop words, like "a," "of," etc.
  - Fixing misspelled words.
  - Reducing words to their stem (e.g. "play" from "playing") using stemming or lemmatization algorithms.

N-grams

- A more sophisticated approach is to create a vocabulary of grouped words. This both changes the scope of the vocabulary and allows the bag-of-words to capture a little bit more meaning from the document.
- in this approach, each word or token is called a “gram”. Creating a vocabulary of two-word pairs is, in turn, called a bigram model. Again, only the bigrams that appear in the corpus are modeled, not all possible bigrams.
- “An N-gram is an N-token sequence of words: a 2-gram (more commonly called a bigram) is a two-word sequence of words like “please turn”, “turn your”, or “your homework”, and a 3-gram (more commonly called a trigram) is a three-word sequence of words like “please turn your”, or “turn your homework”.”

N-grams

- For example, the bigrams in the first line of text in the previous section: “It was the best of times” are as follows:
  - “it was”
  - “was the”
  - “the best”
  - “best of”
  - “of times”
- A vocabulary then tracks triplets of words is called a trigram model and the general approach is called the n-gram model, where n refers to the number of grouped words.
- Often a simple bigram approach is better than a 1-gram bag-of-words model for tasks like documentation classification.

N-grams

Sentence: 2-grams:	The cat sat on the mat the-cat, cat-sat, sat-on, on-the, the-mat				
Notice how even these short n-grams “make sense” as linguistic units. For the other sentence we would have different features:					
Sentence:	The	mat	sat on the	cat	
2-grams:	the-mat, mat-sat, sat-on, on-the, the-cat				
We can go still further and construct 3-grams:					
Sentence:	The cat	sat	on the	mat	
3-grams:	the-cat-sat, cat-sat-on, sat-on-the, on-the-mat				
Which capture still more of the meaning:					
Sentence:	The	mat	sat on	the	cat
3-grams:	the-mat-sat, mat-sat-on, sat-on-the, on-the-cat				

Scoring Words

- Once a vocabulary has been chosen, the occurrence of words in example documents needs to be scored.
- In the worked example, we have already seen one very simple approach to scoring: a binary scoring of the presence or absence of words.
- Some additional simple scoring methods include:
    - **Counts**. Count the number of times each word appears in a document.
    - **Frequencies**. Calculate the frequency that each word appears in a document out of all the words in the document.
- Counts : It was the best of times and worst of times = [1,1,1,1,2,2,1,0,0,0]
- Frequencies: there are 10 words in the document [0.1,0.1,0.1,0.1,0.2,0.2,0.1,0,0,0]

Limitations of Bag-of-Words

- The bag-of-words model is very simple to understand and implement and offers a lot of flexibility for customization on your specific text data.
- It has been used with great success on prediction problems like language modeling and documentation classification.
- Nevertheless, it suffers from some shortcomings, such as:
  - **Vocabulary**: The vocabulary requires careful design, most specifically in order to manage the size, which impacts the sparsity of the document representations.
  - **Sparsity**: Sparse representations are harder to model both for computational reasons (space and time complexity) and also for information reasons, where the challenge is for the models to harness so little information in such a large representational space.
  - **Meaning**: Discarding word order ignores the context, and in turn meaning of words in the document (semantics). Context and meaning can offer a lot to the model, that if modeled could tell the difference between the same words differently arranged (“this is interesting” vs “is this interesting”), synonyms (“old bike” vs “used bike”), and much more.

Term Frequency

- The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- We want to use t_f when computing query-document match scores.
- But how?
- Raw term frequency is not what we want because:
- A document with $t_f = 10$ occurrences of the term is more relevant than a document with $t_f = 1$ occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

- The number of times a word appears in a document divded by the total number of words in the document. Every document has its own term frequency.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Frequency in document vs. frequency in collection

- In addition, to term frequency (the frequency of the term in the document) . . . . . we also want to use the frequency of the term in the collection.
- How many frequent vs. infrequent terms should we expect in a collection?
  In natural language, there are a few very frequent terms and very many very rare terms.
- cfi is collection frequency: the number of occurrences of the term ti in the collection.

- Rare terms are more informative than frequent terms.
  Consider a term that is rare in the collection
  (e.g., arachnocentric).
  A document containing this term is very likely to be relevant.
  → We want high weights for rare terms like arachnocentric.
- Frequent terms are less informative than rare terms.
  Consider a term in the query that is frequent in the collection
  (e.g., good, increase, line).
  A document containing this term is more likely to be relevant
  than a document that doesn't . . .
  . . . but words like good, increase and line are not sure
  indicators of relevance.
  → For frequent terms like good, increase, and line, we
  want positive weights . . .
  . . . but lower weights than for rare terms

Document frequency

- We want high weights for rare terms like arachnocentric.
- We want low (positive) weights for frequent words like good, increase, and line.
- We will use document frequency to factor this into computing the matching score.
- The document frequency is the number of documents in the collection that the term occurs in.

Inverse Data Frequency (IDF)

- dft is the document frequency, the number of documents that t occurs in.
- dft is an inverse measure of the **informativeness** of term t.
- We define the idf weight of term t as follows:

$$idf(w) = log(\frac{N}{df_t})$$

- The log of the number of documents divided by the number of documents that contain the word ***w***. Inverse data frequency determines the weight of rare words across all documents in the corpus.

Example

- Let's try running through an example of computing TF-IDF in action.
- Suppose our search engine contained the following three documents (a.k.a. three webpages):
  - **Document A:** "the mouse played with the cat"
  - **Document B:** "the quick brown fox jumped over the lazy dog"
  - **Document C:** "dog 1 and dog 2 ate the hot dog"
- We compute the *TF-IDF vector* for each document like so:

- **Document A:** "the mouse played with the cat“

$$\text{Relevance}(\text{the}, \text{Doc}_A) = \frac{2}{6} \times \ln\left(\frac{3}{3}\right) = 0.0$$

$$\text{Relevance}(\text{mouse}, \text{Doc}_A) = \frac{1}{6} \times \ln\left(\frac{3}{1}\right) = 0.183102$$

$$\text{Relevance}(\text{played}, \text{Doc}_A) = \frac{1}{6} \times \ln\left(\frac{3}{1}\right) = 0.183102$$

$$\text{Relevance}(\text{with}, \text{Doc}_A) = \frac{1}{6} \times \ln\left(\frac{3}{1}\right) = 0.183102$$

$$\text{Relevance}(\text{cat}, \text{Doc}_A) = \frac{1}{6} \times \ln\left(\frac{3}{1}\right) = 0.183102$$

- So, the final dictionary will look like {"the"=0.0, "mouse"=0.183102, "played"=0.183102, "with"=0.183102, "cat"=0.183102}

- Document **B:** "the quick brown fox jumped over the lazy dog"

$$\text{Relevance}(\text{the}, \text{Doc}_B) = \frac{2}{9} \times \ln\left(\frac{3}{3}\right) = 0.0$$

$$\text{Relevance}(\text{quick}, \text{Doc}_B) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

$$\text{Relevance}(\text{brown}, \text{Doc}_B) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

$$\text{Relevance}(\text{fox}, \text{Doc}_B) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

$$\text{Relevance}(\text{jumped}, \text{Doc}_B) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

$$\text{Relevance}(\text{over}, \text{Doc}_B) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

$$\text{Relevance}(\text{lazy}, \text{Doc}_B) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

$$\text{Relevance}(\text{dog}, \text{Doc}_B) = \frac{1}{9} \times \ln\left(\frac{3}{2}\right) = 0.045052$$

- Document C: “dog 1 and dog 2 ate the hot dog"

$$\text{Relevance}(\text{dog}, \text{Doc}_C) = \frac{3}{9} \times \ln\left(\frac{3}{2}\right) = 0.135155$$

$$\text{Relevance}(1, \text{Doc}_C) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

$$\text{Relevance}(\text{and}, \text{Doc}_C) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

$$\text{Relevance}(2, \text{Doc}_C) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

$$\text{Relevance}(\text{ate}, \text{Doc}_C) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

$$\text{Relevance}(\text{the}, \text{Doc}_C) = \frac{1}{9} \times \ln\left(\frac{3}{3}\right) = 0.0$$

$$\text{Relevance}(\text{hot}, \text{Doc}_C) = \frac{1}{9} \times \ln\left(\frac{3}{1}\right) = 0.122068$$

- From the above table, we can see that TF-IDF of common words was zero, which shows they are not significant. On the other hand, the TF-IDF of “ate” , “for”, “dog”, and “cat” are non-zero. These words have more significance.

Document as vectors

- Each document is now represented as a real-valued vector of tf-idf weights $\in R|V|$.
- So we have a $|V|$-dimensional real-valued vector space.
- Terms are axes of the space.
- Documents are points or vectors in this space.
- Very high-dimensional: tens of millions of dimensions when you apply this to web search engines
- Each vector is very sparse - most entries are zero.