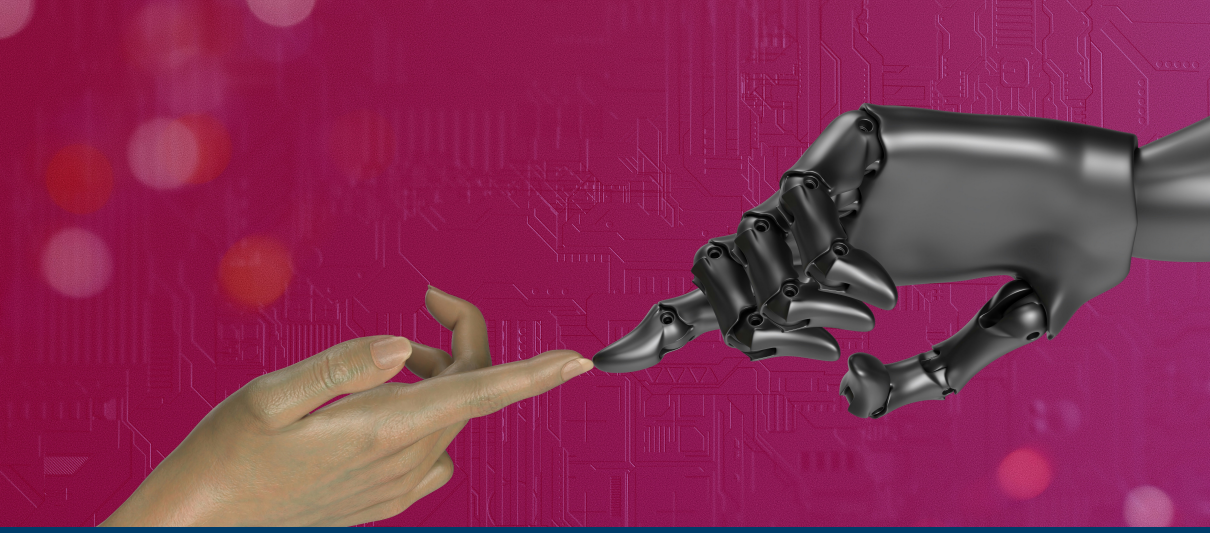

Hands-On Machine Learning

Dr. Derek Bridge | 2026/2027

UCC Coláiste na hOllscoile Corcaigh
University College Cork, Ireland

School of Computer Science and Information Technology
Scoil na Ríomheolaíochta agus na Teicneolaíochta Faisnéise

13. Text Preprocessing

In this lecture, our dataset comprises just three social media posts.

Structured Dataset

- Each example fits a predefined schema.
- Most common is rows and columns, hence also called **tabular datasets**.

Unstructured datasets

- Examples are in their native format.
- E.g. a dataset of texts, or images, or audio, or video.

Text Datasets

- A **text dataset** is also sometime called a **corpus**.
- Each *example* is sometimes referred to as a **document** — whether it be as short as a social media post/ SMS/ email or as long as a student essay.

Text Preprocessing

- **Text Preprocessing** converts examples from raw text into a more structured, numeric format that can be handled by Machine Learning algorithms.
- We study **Bag-Of-Words**, where each document becomes a vector of numbers.

Sets

- A **set** is a collection of objects with two properties:
  - Order is not important, e.g. $\{a, c, f\} = \{c, f, a\}$
  - Duplicates are not allowed, e.g. $\{a, c, a, f\} = \{a, c, f\}$
- The set of all possible objects U is called the **universal set**, e.g. $U = \{a, b, c, d, e, f, g\}$

Data Structures for Sets

- There are many data structures we can use to store sets, e.g. linked lists, binary search trees, ...
- But we can describe a set by a binary-valued vector.
- E.g. if $U = \{a, b, c, d, e, f, g\}$, then we can represent the set $\{a, c, f\}$ as follows:

a	b	c	d	e	f	g
1	0	1	0	0	1	0

- Then, we can store the vector as, e.g., an array.
- In fact, if U is large but the sets are much smaller, then our vectors will be **sparse** (mostly zero).
- It may be more efficient to use a data structure that only stores the non-zero elements.

Bags

- A **bag** is a collection of objects with one property:
  - Order is not important, e.g. $\{a, c, f\} = \{c, f, a\}$
  - This time, duplicates *are* allowed, e.g. $\{a, c, a, f\} \neq \{a, c, f\}$

Data Structures for Bags

- We can describe a bag by an integer-valued vector, where the numbers are the frequencies with which the elements occur.
- E.g. if $U = \{a, b, c, d, e, f, g\}$, then we can represent the bag $\{a, c, a, f\}$ as follows:

a	b	c	d	e	f	g
2	0	1	0	0	1	0

- We can store these as arrays or sparse arrays.

Bag-Of-Words

- We will represent each document in our corpus (each example in our text dataset) as a **bag of words**.
- This will lose lots of information. Start thinking about what we will lose!
- Nevertheless, Bag-Of-Words is very competitive on text classification tasks such as spam filtering and sentiment analysis.

Running Example

A dataset comprising three Barack Obama social media posts (quoting Nelson Mandela)

"No one is born hating another person because of the color of his skin or his background or his religion."
"People must learn to hate, and if they can learn to hate, they can be taught to love."
"For love comes more naturally to the human heart than its opposite."

Text Preprocessing

- Tokenization.
- Optionally, discard stop-words.
- Optionally, apply stemming or lemmatization.
- Either count vectorization or TF-IDF vectorization.

Tokenization

- First, we must **tokenize** each document. This means splitting it into **tokens** (sometimes also called **terms**).
- Simplest approach: change uppercase to lowercase, treat punctuation symbols as spaces, split at the spaces.
- Roughly, this mean one token = one word.

Discussion

- In reality, tokenization is surprisingly complicated.
- E.g. should we treat "People" and "people"' as different tokens?
- E.g. should we keep the punctuation as separate tokens?
- E.g. is "don't" one token or two or three?
- E.g. should **bigrams** (pairs of consecutive words) become tokens?

Running Example

{"no", "one", "is", "born", "hating", "another", "person", "because", "of", "the", "color", "of", "his", "skin", "or", "his", "background", "or", "his", "religion"}
{"people", "must", "learn", "to", "hate", "and", "if", "they", "can", "learn", "to", "hate", "they", "can", "be", "taught", "to", "love"}
{"for", "love", "comes", "more", "naturally", "to", "the", "human", "heart", "than", "its", "opposite"}

Stop-words

- Optionally, we can discard **stop-words**.
- Typically, stop-words are common words such as "a", "the", "in", "on", "is", "are",...
- (Linguists call these the closed-class words or function words or grammatical words —categories of words, such as articles and prepositions, that do not readily accept new members— as opposed to the open-class words or content words or lexical words —categories of words, such as nouns, verbs, adjectives and adverbs, where it is easier to add new words.)

Discussion

- For simple classifiers (e.g. spam filters, sentiment analysers), discarding stop-words does no harm.
- Other times, we would lose too much, e.g. consider discarding them from queries to a web search engine where the query is "to be or not to be".

Running Example

{"born", "hating", "person", "color", "skin", "background", "religion"}
{"people", "learn", "hate", "learn", "hate", "taught", "love"}
{"love", "comes", "naturally", "human", "heart", "opposite"}

Stemming

- Optionally, we can **stem** the words.
- Stemming uses rules-of-thumb to remove or replace prefixes or suffixes, so that variants of a word are reduced to the same stem form.
- E.g. drop "ing", "s", "ly", "un", ...

Lemmatization

- Also optional, **lemmatization** is an alternative to stemming.
- It reduces variants of a word to the base or root form of the word, as it appears in a dictionary.

Discussion

- Stemming/lemmatization reduce dimensionality and may reveal commonalities among texts that were originally different.
- Lemmatization is more expensive than stemming.

Running Example: Stemming (by NLTK's rules)

{"born", "hate", "person", "color", "skin", "background", "religion"}
{"peopl", "learn", "hate", "learn", "hate", "taught", "love"}
{"love", "come", "natur", "human", "heart", "opposit"}

Running Example: Lemmatization

{"born", "hate", "person", "color", "skin", "background", "religion"}
{"people", "learn", "hate", "learn", "hate", "teach", "love"}
{"love", "come", "natural", "human", "heart", "opposite"}

Count Vectorization

- Each document (example) becomes a vector.
- Each token becomes a feature.
- Feature-values are **frequencies** — how many times that token appears in that document.

Running Example

back-ground	born	color	come	hate	heart	human	learn	love	natur	opposit	peopl	person	religion	skin	taught
1	1	1	0	1	0	0	0	0	0	0	0	1	1	1	0
0	0	0	0	2	0	0	2	1	0	0	1	0	0	0	1
0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0

TF-IDF Vectorization

- **TF-IDF Vectorization** is an alternative to Count Vectorization.
- The feature-values are now **tf-idf scores** — they penalize words that recur across multiple documents on the principle that they are less discriminative.
- E.g. in emails, words such as “hi”, “best”, “regards” will be penalized.

Running Example

back-ground	born	color	come	hate	heart	human	learn	love	natur	opposit	peopl	person	religion	skin	taught
0.39	0.39	0.39	0	0.30	0	0	0	0	0	0	0	0.39	0.39	0.39	0
0	0	0	0	0.51	0	0	0.67	0.26	0	0	0.34	0	0	0	0.34
0	0	0	0.42	0	0.42	0.42	0	0.32	0.42	0.42	0	0	0	0	0

TF-IDF

- Simplest version:

$$tf\_idf(t, d) = tf(t, d) \times idf(t, D)$$

where $tf(t, d)$ is **term frequency**

$$tf(t, d) = \frac{\text{number of times token } t \text{ appears in document } d}{\text{total number of tokens in document } d}$$

and $idf(t, D)$ is **inverse document frequency**

$$idf(t, D) = \log \frac{\text{total number of documents in corpus } D}{\text{number of documents containing token } t}$$

- Variants of the formula might: logarithmically scale tf to avoid biases towards long documents; add 1 to parts of the formula to avoid edge cases; normalize the results, e.g. divide by the l_2-norm.

Unigrams

- We have assumed **unigrams**: one token = one word.
- E.g. "No one is born hating another person because of the color of his skin or his background or his religion."
- The tokens are "no", "one", "is", "born", ...

Bigrams

- We could additionally use **bigrams**: one token = a pair of consecutive words.
- E.g. in addition to the unigrams, the tokens are "no one", "one is", "is born"', "born hating", ...

Trigrams

- We could additionally use **trigrams**: one token = sequences of three consecutive words.
- E.g. in addition to the unigrams and bigrams, the tokens are "no one is", "one is born", "is born hating"', ...
- In general, we refer to *n*-**grams**.

- The cost is many more features (tokens) — vectors of even higher dimensionality and sparsity.
- The advantage is now we are making available to the classifier some of the grammatical structure (word order, the functions of the stop-words, etc) — we lose it if we use only unigrams.

Task-Specific Tokens (Hacks!)

Spam Filtering

- Words with punctuation inside them, e.g. “loan$”, “mort.gage”, “vi@gra”, can all be mapped to a single token – `SPECIAL_TOKEN`.
- Tell-tale phrases can be treated as single tokens, e.g., “submit your manuscript”, “urgent reply” — even when you are not using *n*-grams more generally.

Sentiment Analysis

- Words preceded by “not” can be replaced by a single token, e.g. “not like” and “not recommend” become `NOT_like` and `NOT_recommend` — even when you are not using *n*-grams more generally.
- Emoticons can be left as tokens — even when other punctuation is being stripped. E.g. `:)` and `:=(`

Language Identification

- Use character bigrams and trigrams as tokens — instead of words.
- E.g. tokens from English: “goo”, “ood”, “od ”, “d l”, “ lu”, “luc”, “uck”.
- E.g. tokens from mystery language: “pow”, “owo”, “wod”, “odz”, “dze”, “zen”, “eni”, “nia”.