# Session 09: NLP basics

wifi: GA-Guest, yellowpencil

GENERAL ASSEMBLY

```
cd ~/Documents/ga-ldn-ds37
git commit -am “your commit message here”
git pull
```

 

# Today’s session plan

| | |
|---|---|
| **1800-1810** | Standup |
| **1830-1900** | Linear algebra review |
| **1900-1920** | Break |
| **1920-2100** | Project troubleshooting |
| | |

# At the end of the session, you will be able to ...

**Vectorise** text using count vectorisers and TF-IDF

**Use** pre-processing methods including stemming and removing stopwords

**Implement** text classification with Naive Bayes

**Develop** your independent project work further

Data Science Part Time

# What is NLP?

# Natural language processing

Natural language processing is a field concerned with:

- Using computers to process human language and text
- Making sense of human knowledge stored as unstructured text

Natural language processing has many applications in data science, including:

- Text classification
- Topic modelling or clustering
- Translation
- Text summarisation or simplification
- Sentiment analysis
- Speech to text
- Text to speech

 

# Natural language processing

Many natural language processing tasks will involve the following pre-processing techniques:

**Tokenization** Breaking a sentence into words

**Stopword removal** Removing and/it/they etc and other grammatically non-useful words

**Stemming and lemmatization** Reducing words back to their original roots

**Part of speech tagging** Labelling nouns, verbs, adjectives

**Vectorisation** Representing text data as a numerical vector

# Why is natural language processing hard?

Here are just a few of the reasons why NLP is challenging:

**Ambiguity**

**Non standard English**

**Tricky entity names**

**Newly coined words**

**Idioms**

 

**Computers Out:**

# Getting started with NLP

Now let's open ds37-09-01.ipynb to get started

# Some Terminology

**Document**

A single observation in a dataset. This could be a single review in a database of customer reviews, a single tweet in a set of tweets, etc

**Corpus**

A collection of documents

**Token**

A single word in a document. Sometimes a token can be a single phrase, of a specified length. For example, we might want to split a document into two-word tokens.

 

# Stopwords

Stop words are some of the most common words in a language. They are used so that a sentence makes sense grammatically, such as prepositions and determiners, e.g., "to," "the," "and." However, they are so commonly used that they are generally worthless for predicting the class of a document.

```
> stopwords("english")
 [1] "i"          "me"         "my"          "myself"     "we"
 [6] "our"        "ours"       "ourselves"   "you"        "your"
[11] "yours"      "yourself"   "yourselves"  "he"         "him"
[16] "his"        "himself"    "she"         "her"        "hers"
[21] "herself"    "it"         "its"         "itself"     "they"
[26] "them"       "their"      "theirs"      "themselves" "what"
[31] "which"      "who"        "whom"        "this"       "that"
[36] "these"      "those"      "am"          "is"         "are"
[41] "was"        "were"       "be"          "been"       "being"
[46] "have"       "has"        "had"         "having"     "do"
```

 

# Stemming and lemmatization

These are both ways of intelligently reducing the number of features by grouping together (hopefully) related words.

**Stemming** is the process of removing common endings from sentences, such as "s", "es", "ly", "ing", and "ed". This reduce a word to its base/stem/root form.

**Lemmatization** is a more refined process that uses specific language and grammar rules to derive the root of a word. This is useful for words that do not share an obvious root such as "better" and "best".

| Lemmatization | Stemming |
|---|---|
| shouted → shout | badly → bad |
| best → good | computing → comput |
| better → good | computed → comput |
| good → good | wipes → wip |
| wiping → wipe | wiped → wip |
| hidden → hide | wiping → wip |

 

# Bag of words model

Converts a corpus of text to a **term-document matrix**, where every row corresponds to a **document** and every column is a feature or **unique word**.

The value of each element in the matrix is either a binary indicator, marking the presence of that word in the document, or a word count.

**How can we convert a corpus of text to a matrix?**

| | it | is | puppy | cat | pen | a | this |
|---|---|---|---|---|---|---|---|
| it is a puppy | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| it is a kitten | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| it is a cat | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| that is a dog and this is a pen | 0 | 2 | 0 | 0 | 1 | 2 | 1 |
| it is a matrix | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

# Vectorising

There are a few different ways of vectorising text documents. One simple method is **count vectorising.** Each matrix entry is the count of a particular word in a particular document.

| | it | is | puppy | cat | pen | a | this |
|---|---|---|---|---|---|---|---|
| it is a puppy | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| it is a kitten | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| it is a cat | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| that is a dog and this is a pen | 0 | 2 | 0 | 0 | 1 | 2 | 1 |
| it is a matrix | 1 | 1 | 0 | 0 | 0 | 1 | 0 |

**Solo Exercise:**

# Count vectorising

Take the following sentences and go through the steps below, by hand:

The cat sat on the mat
The rat was sat on the cat
The mat is red
The cat is black

1. Compile the **vocabulary** for this corpus. That's the set of unique words across the whole corpus, with no repeat words.
2. On paper, compile the term-document matrix for this corpus.

# Naive Bayes

Once we've vectorised our text, we use Bayes' theorem to learn the relationship between a document's features (i.e. the words in it) and its label.

Bayes' Theorem, is a way to assess probabilities, using prior information to make more accurate predictions.

$$P(A \mid B) = \frac{P(B \mid A) \times P(A)}{P(B)}$$

- $P(A \mid B)$ : Probability of `Event A` occurring given `Event B` has occurred.
- $P(B \mid A)$ : Probability of `Event B` occurring given `Event A` has occurred.
- $P(A)$ : Probability of `Event A` occurring.
- $P(B)$ : Probability of `Event B` occurring.

 

**Solo Exercise:**

# Bayes' theorem

At a GP surgery, 10% of patients are prescribed painkillers.

Overall, five percent of the surgery's patients are addicted to painkillers.

Out of all the people prescribed pain pills, 8% are addicts.

If a patient is an addict, what is the probability that they will be prescribed pain pills?

# Bayes' theorem

**Step 1** Figure out what your event "A" is from the question. That information is in the italicized part of this particular question. The event that happens first (A) is being prescribed pain pills. That's given as 10%.

**Step 2** Figure out what your event "B" is from the question. That information is also in the italicized part of this particular question. Event B is being an addict. That's given as 5%.

**Step 3** Figure out what the probability of event B (Step 2) given event A (Step 1). In other words, find what (B|A) is. We want to know "Given that people are prescribed pain pills, what's the probability they are an addict?" That is given in the question as 8%, or .8.

**Step 4** Insert your answers from Steps 1, 2 and 3 into the formula and solve.

$P(A|B) = P(B|A) * P(A) / P(B) = (0.08 * 0.1)/0.05 = 0.16$

The probability of an addict being prescribed pain pills is 0.16 (16%).

 

# Bayes' theorem

How does Bayes' theorem apply to our text classification problem? Imagine we're trying to classify emails into two classes: 'spam' or 'ham'. Our first email contains just three words: 'send money now' so the probability that it's spam is as follows

$$P(spam \mid \text{send money now}) = \frac{P(\text{send money now} \mid spam) \times P(spam)}{P(\text{send money now})}$$

We assume our features are **conditionally independent** so our calculation simplifies to:

We know what each of these terms is from the training data

$$P(spam \mid \text{send money now}) \approx \frac{P(\text{send} \mid spam) \times P(\text{money} \mid spam) \times P(\text{now} \mid spam) \times P(spam)}{P(\text{send money now})}$$

 

# Bayes' Theorem

Imagine we plug these numbers in from our training data:

$$P(spam \mid \text{send money now}) \approx \frac{0.2 \times 0.1 \times 0.1 \times 0.9}{P(\text{send money now})} = \frac{0.0018}{P(\text{send money now})}$$

Now we repeat the calculation for our **other** class, ham:

$$P(ham \mid \text{send money now}) \approx \frac{0.05 \times 0.01 \times 0.1 \times 0.1}{P(\text{send money now})} = \frac{0.000005}{P(\text{send money now})}$$

All we care about is which class has a higher probability, so we can ignore the denominator and pick the highest probability; in this case that's **spam**.

# Bayes' Theorem

The key takeaways from Bayes' are:

- The "naive" assumption of Naive Bayes (that the features are conditionally independent) is critical to making these calculations simple.
- The normalization constant (the denominator) can be ignored since it's the same for all classes.

**Advantages of Naive Bayes**
Model training and prediction are very fast.
It's somewhat interpretable.
No fine tuning is required.

**Disadvantages of Naive Bayes**
If "spam" is dependent on non-independent combinations of individual words, it may not work well.
Correlated features can be problematic (due to the independence assumption).

# Conditional independence

The events $R$ and $B$ are conditionally independent [given $Y$] if and only if, given knowledge of whether $Y$ occurs, knowledge of whether $R$ occurs provides no information on the likelihood of $B$ occurring, and knowledge of whether $B$ occurs provides no information on the likelihood of $R$ occurring.

**For example**

Let the two events be the probabilities of persons A and B getting home in time for dinner, and the third event is the fact that a snow storm hit the city. While both A and B have a lower probability of getting home in time for dinner, the lower probabilities will still be independent of each other. That is, the knowledge that A is late does not tell you whether B will be late. (They may be living in different neighborhoods, traveling different distances, and using different modes of transportation.)

 

# TF-IDF

There are alternative methods for vectorising text; one of these is TF-IDF.

TF-IDF analyses the uniqueness of words between documents to find distinguishing characteristics.

Term frequency–inverse document frequency (TF–IDF) computes the "relative frequency" with which a word appears in a document, compared to its frequency across all documents.

It's more useful than "term frequency" for identifying "important" words in each document (high frequency in that document, low frequency in other documents). The TF-IDF score for a given word in a given document is:

$$\frac{\text{number of times the word is in this document}}{\text{number of documents the word is in}}$$

**Solo Exercise:**

# TF-IDF

Given the following documents, compile a term-document matrix using TF-IDF. **Before** you start, which word do you think will have the highest TF-IDF score? Why?

The cat and dog sat
The dog and cat sat
The cat sat and sat
The cat killed the dog

Intro to Python

# Let’s Review

# At the end of the session, you will be able to ...

**Vectorise** text using count vectorisers and TF-IDF

**Use** pre-processing methods including stemming and removing stopwords

**Implement** text classification with Naive Bayes

**Develop** your independent project work further

 

# Coming up next time...

- Linear regression