1 Introduction to Text Mining

Text mining is the discipline that tries to extract knowledge (structured) from text (unstructured). It uses methods from Natural Language Processing but is not the same as NLP also does translation, speech recognition and semantic parsing. Another discipline, information retrieval, is the first step in text mining processes as to filter the documents to actually mine. But text mining generally does not use approaches like ranking models or user modeling and evaluation.

1.1 Challenges

Some challenges that text mining projects generally face are errors in the text as it might be generated through OCR or ASR, the text might not be labelled or might miss punctuation and capitalisation, different languages might be used in the text and the terms in the text might have synonyms or could be polysyms (multiple meanings and uses for the same word).

- **Unstructured**
  Missing capitalisation, punctuation or unlabelled plain text that has no metadata attached to it whatsoever.
- **Noise**
  The **source** text can contain non-content information (i.e. HTML), encoding errors, page numbers or other noise. **Content** might contain OCR and ASR artifacts, spelling errors, typos or (lost) formatting. **Labelling** might be incomplete or contain wrong labels.
- **Infinity**
  Every new document is likely to introduce new terms, the first moreso (at a very rapid rate) than latter documents (at a slower rate):

  Definition 1.1 (Heaps' Law)**.** As more instance text is gathered, there will be diminishing returns in terms of discovery of the full vocabulary from which the distinct terms are drawn.
- **Ambiguity**
  Phrases might have a different meaning in a different context. For example, 2 stars is poor for a hotel, but excellent in a michelin rating.

1.2 Tasks

There are three main types of text mining tasks: text classification (or clustering), sequence labelling or text-to-text generation.

1.2.1 Classification

Classification (or clustering) assigns labels to a document. The document can be many things, from a sentence or a tweet to entire articles or maybe even a book. The labels can also be many things, such as the topic of the document, relevance, the author of the document, sentiment analysis, et cetera.

In classification, the order of words is not as relevant, and traditionally represent the text as a bag of words. Each word in the collection becomes a machine learning feature. This results in high-dimensional sparse vectors (i.e. dimension is the distinct words, and the values are counts).

Definition 1.2 (Bag of words approach)**.** Each word in a document is counted without punctuation, capitalization or order. A vector with every word in the corpus as the dimension and the respective count in the document as values represents the text. For example: The brown fox $\rightarrow$ [the, brown, some, fox] $\rightarrow [1, 1, 0, 1]$.

Only very few words are very frequent in documents. There is an strong inverse relation between the word frequency and the word rank, also called the long tail distribution. Words in the tail might be noise or errors and we want to get rid of them.

Definition 1.3 (Zipf's law)**.** When a list of measured words is sorted in decreasing order, the value of the nth entry is often approximately inversely proportional to n.

An alternative representation of words that is dense and has a lower dimension is word embeddings. Here words are represented in the vector space close to relevant words.

1.2.2 Sequence labelling

Sequence labelling identifies named entities in a text (such as persons, places, actions). Here the order matters, punctuation and capitalization is important.

1.2.3 Text-to-text generation

Text is given as an input, and resulting text is generated as an output. Use cases are things like summarization, translation.

1.3 Transformers

Transformers take an input text and generate an output text, they consist of two parts, an encoder that processes the input text and a decoder that generates the output text. Initially they were designed for translation tasks and were used in Google Translate.

BERT is a transformer model that only gives the encoder, providing word embeddings for input text. GPT is a decoder-only transformer, where the input is a text (or a prompt) and the output is text.

- **Encoder** (BERT)
  Classification tasks, sequence labelling.
- **Decoder** (GPT)
  Text generation (i.e. autocomplete).
- **Encoder-decoder** (T5)
  Summarization and translation.

1.4 Paradigms

- **Supervised Learning**
  Train feature-based models (i.e. bag of words in SVM). Lightweight and explainable, important to understand the model.
- **Transfer Learning**
  Use a pre-trained model and fine-tune it for a task. Best option when there is sufficient labelled data.
- **In-context Learning**
  Prompt a large language model with instructions and examples. Can be used without labelled data or undefined outcomes.

1.5 Quiz

Quiz 1.1 (If we use words as features in a text classification task, the resulting vectors are)**.**
High-dimensional and sparse.

Quiz 1.2 (What does the long-tail distribution for text data refer to?)**.**
In a given collection, there are many terms with a low frequency and a few terms with a high frequency.

Quiz 1.3 (For which type of text processing task are capitalization and punctuation more useful?)**.**
Sequence labelling

2 Preprocessing

All text needs a clean-up of some kind, either the documents might have encoding errors, the text might be scanned (through OCR) or recognized (from speech). Digital input is also not always clean, as it might contain things like layout, disclaimers, headers, tables, or markup. We want to get rid of this data before we use it in models, but it might contain useful information (i.e. in XML or JSON) that we can use.

2.1 Encoding

Text is encoded for storage, where characters are stored as numbers. Classically this used ASCII but it only encodes simple alphanumeric characters that are used in American English. A universal and independent standard for encoding is Unicode, and rendering of the characters is implemented by the software. A popular implementation of Unicode (UTF-8).

2.2 Cleaning

When we do a text mining task we generally have to start with the creation of a dataset for our task. This means that we likely have to digitize, convert and clean textual content to run our mining task on and can take the majority of the work of the task.

2.2.1 Regular Expressions

One way to get relevant content from textual data is to use regular expressions, these are expressions that can be used to search text. For example, a Dutch postal code is `/[1-9][0-9]3 [A-Z]2/`. These expressions can be used to either extract information from texts, or to remove terms or private data from texts.

However, these might get too complex as the expressions can also start hitting (parts) of words, you might need to account for capitalization or different ways of writing a term.

2.2.2 Spacy

An alternative for Regular Expressions is to use Spacy, which converts words in the text to tokens and then you can simply match on tokens (see next section).

2.3 Tokenization

Text can be converted to tokens for comparison with oneanother, as you might have different ways of writing the same word. For example, `isn't it?` can be written as `is not it?` and contains four tokens (three words, one punctuation).

Definition 2.1 (Token)**.** An instance of a word or punctuation occurring in a document

Definition 2.2 (Word Type)**.** A distinct word in the collection (without duplicates)

Definition 2.3 (Term)**.** A token when used as a feature (or index), generally in normalized form.

Definition 2.4 (Token count)**.** Number of words in a collection/document, includes duplicates.

Definition 2.5 (Vocabulary size)**.** Number of unique terms (word types); the feature size or dimension size for bag-of-words.

The choice of the terms (i.e. whether to use or not to use punctuation) depends on the task that you are trying to complete. For example, in forensics, you might want to specifically look at the use of punctuation.

2.3.1 Sentence Splitting

Besides splitting on tokens, we might need to split text into sentences. For example, when we look at relations between multiple entities in the same sentence. Or, if we want to look at the sentiment of each sentence (such as in reviews). This can be difficult due to markup being used (bullet points), abbreviations, different uses of punctuation or line breaks in the document.

2.3.2 Sub-word Tokenization

Each new document will add new terms, that might relate to words that we used earlier. For example, the words model, models and modeling all have the same "model" root. If we split these words into subwords we can match on *parts* of the words that we have seen before. One of the sub-word tokenizers is byte-pair encoding, which works as follows:

Definition 2.6 (Byte-pair encoding (BPE))**.** Create a vocabulary of all characters. Merge the most occurring adjacent characters to form a new token and apply the new token. Repeat and create new subwords this way until we have k novel tokens.

2.3.3 Lemmatization and Stemming

We might want to normalize words that are written down differently to the same term. For example, think, thinking and thought all have the same meaning in a different tense. In this case you can take the **Lemma** or the **Stem**.

Definition 2.7 (Lemma)**.** The dictionary form of a word, i.e. the infinitive for a verb (thought → think) and the singular for a noun (mice → mouse).

Definition 2.8 (Stem)**.** The prefix of a word, i.e. computer, computing, computers, compute all share `comput`.

We mostly prefer lemmas over stems, as they can merge more different presentations of the text to the same terms.

2.4 Edit Distance

For spelling correction and normalization we can look at the "closest" correct word to the word that was written. The metric that we can use to calculate this is by Levenshtein distance, where insertion, deletion and substitution all have a cost of 1. The match with the lowest cost in that case has the lowest edit distance. You can extend this cost function by for example lowering the costs of characters that are next to eachother on a keyboard or are phonetically similar.

2.5 Quiz

Quiz 2.1 (What is optical character recognition (OCR)?)**.**
A technique for converting the image of a printed text to digital text.

Quiz 2.2 (What is the main limitation of ASCII?)**.**
It can only encode letters that occur in the American English alphabet.

Quiz 2.3 (What does the RegEx match?)**.**
Regular Expression: `/<[^>]+>/`
Any HTML/XML tag.

Quiz 2.4 (Can you make a language-independent lemmatizer?)**.**
No, because a lemmatizer uses a dictionary.

Quiz 2.5 (What is the levenshtein distance between shrimp and shrop?)**.**
2

3 Vector Semantics

3.1 Vector Space Model

In the bag of words approach, we can represent documents and queries are in a vector space with the words as dimensions. Each document can be represented by a vector in this space.

The problem here is that the documents result in very sparse vectors with a very high dimensionality.

The alternative is to not present documents as words but rather by: topics or word embeddings. Topics are discussed in lecture 9.

3.2 Word Embeddings

The basic idea behind representing words as embeddings is that:

Definition 3.1 (Distributional Hypothesis)**.** The context of a word defines its meaning.

For example, if we have *a bottle of foobar*, *a glass of foobar* and *foobar gets you really drunk*, then it is likely that foobar is an alcoholic beverage. Meaning, that a word can be assigned a value by the words that surround it, and words with similar values are likely similar words.

The idea is to create a dense vector space where we place words that are similar close to eachother. The dimensionality is between 100 to 400 here, which is low dimension in NLP terms but rather high in other disciplines. The similarity between the words is learned, not from lemmas. The words are mapped to syntactically and semantically similar words in a continuous dense vector space using the distributional hypothesis.

These embeddings are generated through feed-forward neural networks with the vector dimension as the number of output nodes and the probabilities in the output are normalized using the softmax function (which will exaggerate the differences and give a clear choice from the vector).

3.2.1 Word2Vec

Word2Vec is an early efficient predictive model to learn the weights for word embeddings. However superseded by BERT, it is still used regularly as it is very efficient. It has a single hidden layer, so it is not a deep neural network, and searches for the probability of words are likely to show up near another word.

After training, the hidden layer then has the weights of each word respective (completely connected) to the other words. We start with the document, extract a vocabulary and try to represent it as a lower dimensional vector.

This is a supervised task, but we did not label the data ourselves meaning it is a self-supervised task (or approach). Word2Vec does this by applying skip-gram with negative sampling, which means that it:

1. Take a target word, and a neighbouring context window as positive examples
2. Randomly sample other words as negative samples
3. Train a classifier (with one hidden layer) to distinguish these two cases
4. The learned weights of the hidden layer are the embeddings
5. Update previous embeddings of the word if necessary

It scales well, learned embeddings can be re-used, and training can be done incrementally. However, embeddings are static, meaning the same word always has the same "meaning". Embeddings can be used as input for classifiers but can't be updated while training the classifier.

3.3 Document Embeddings

It might also be useful to generate embeddings for documents, as to create vectors from documents. For example, you could take the geometric average of all words in the document, but many documents will end up close to the center in this case.

An alternative is to use `doc2vec` where it generates a separate embedding for each paragraph in the document takes the words that it co-occurs with as a positive and words that it doesn't as negative. This can then be used as an input to a classifier.

3.4 Quiz

Quiz 3.1 (What is the role of the distributional hypothesis in training word embeddings?)**.**
Words that occur in similar contexts get similar representations

4 Text Categorization

We can separate classification tasks in three distinct categories: binary classification (yes/no), multi-class classification (a/b/c) and multi-label classification (nil/a/a,b/...).

4.1 Task Definition

4.1.1 Text Unit

First define the text unit, i.e. the size and boundary of the document. For example, complete documents (articles, emails), sections (minutes, speeches) or sentences (language identification, sentiment classification).

4.1.2 Categories

What is the category that we want to extract from the documents, i.e. spam/no spam, relevant/irrelevant, language, sentiment, stance, topic/subtopic (i.e. which type of cancer), warning (i.e. detect hate speech).

4.1.3 Pre-processing

We want to have features for documents, for example a vector from each document with the same dimensionality. Using the bag of words you would have a high-dimensional vector that is very sparse, the advantage is that it is very transparent and you can show from the weights why a classifier learned a specific model. Alternatively, you can use embeddings which is less interpretable.

Using words as features on the vectors x_i and the class label as y_i. Before we use the words as features we tokenize them, and generally we also lowercase and remove punctuation. BERT models come cased which keeps capitalization and diacritical marks, and uncased which removes these. Further steps include, but are not limited to, removing stopwords, lemmatizing or stemming, or adding phrases (i.e. not good) as a single feature.

4.1.4 Feature Selection

Limiting the amount of features reduces the dimensionality and avoids overfitting to the training data. Furthermore, due to the long tail of the word distribution, we only want to use words that appear in multiple documents.

Globally, we can fix the vocabulary size and keep only the top-n most frequent terms. Rare terms can also be avoided by using a cut-off on the term frequency. For example in the CountVectorizer you can set parameters to have only terms that occur in a ratio of max_df documents and at least min_df times.

4.1.5 Term Weighting

We can add the feature weights to the document-term matrix in binary (occurs or not), integer (term count) or a real value (more advanced). Real values can often contain more information than just the counts, a popular weighting scheme is Tf-Idf. In this case there are two components: the term frequency (tf) and inverse document frequency (idf).

In Tf-Idf the term frequency (tf) counts how often the term occurs in the document. Instead of using the raw value, we use a 1 + log value such that the term counts are normalized closer to eachother. Meaning, 1 becomes 1, 10 becomes 2 and 100 becomes 3.

The inverse document frequency (idf) uses the intuition that the most frequent terms are not very informative. It gives the number of documents that the term occurs in and takes the log of the inverse of that.

Definition 4.1 (Tf-Idf)**.** $tf = 1 + log(\text{term}_{\text{count}})$, $idf = log(\frac{|\text{docs}|}{df_{\text{term}}})$, $tf \times idf$.

4.1.6 Classifier Learning

If we use word features, we need an estimator that is well-suited for high-dimensional and sparse data. Such as Naive Bayes, Support Vector Machines or Random Forests. If we have embeddings, we can use transfer learning, see lecture 6. Alternatively, we can use in-context learning with a generative LLM.

Naive Bayes One of the older models is Naive Bayes which has been used for SPAM classification in emails for a long time. It uses the prior probability of each category in the training data and use the posterior probability distribution over the possible categories.

In essence, we take a document and calculate for all classes the probability of that document and that class, times the probability of the class in general, divided by the probability of the document. But, considering that the document probability is a constant given, we can eliminate that term and only compute the probability of the document and the class, times the probability of the class, and take the class where this is the highest.

We can calculate the probability of the document and the class by taking the probability of each term given the class. When we have each term's probability we can multiply the probabilities and get a single probability. This is also what makes this naive, as it assumes that the terms are independent. However, this will fail, as we (almost) always have terms in the document with the probability zero, multiplying the entire probability to zero. This can be solved by adding a smoothing function to the terms. For example, laplace smoothing (add-one smoothing) assumes that each term occurs one additional time.

This approach does the two assumptions that the terms are independent of eachother and assigns the same probabilities to terms regardless of their position in the document. This results in a correct estimation, but inaccurate probabilities, as only the ordering can be used to estimate the correct class.

4.1.7 Evaluation

Split data into a test and training set, for example 80% train and 20% test. Don't train on the test set to prevent overfitting on your dataset. For hyperparameter tuning, use a validation set that is part of your train set, generally using cross validation.

Accuracy is often not suitable as the classes are often unbalanced. High accuracy in one class might mean a low accuracy in the other. It depends on the task what we are interesting in, for example you want a spam filter to mark important emails as not spam more than that you want to mark spam emails as spam.

Generally, precision and recall are used and evaluated on all classes. Where precision measures how many of the assigned labels are correct, and recall is how many of the true labels were assigned. The harmonic mean of the precision and recall is the F1 score which combines the precision and recall.

Definition 4.2 (Precision)**.** $\text{Precision} = \frac{\text{True Positive}}{\text{True Positive}+\text{False Positive}}$

Definition 4.3 (Recall)**.** $\text{Recall} = \frac{\text{True Positive}}{\text{True Positive}+\text{False Negative}}$

Definition 4.4 (F1-score)**.** $\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}}$