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: $\text{The brown fox} \to [\text{the}, \text{brown}, \text{some}, \text{fox}] \to [1, 1, 0, 1]$.

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)**.**

$$\text{tf} = 1 + \log(\text{term}_{\text{count}})$$

$$\text{idf} = \log(\frac{|\text{documents}|}{df_{\text{term}}})$$

$$\text{tf-idf} = \text{tf} \times \text{idf}$$

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.

Definition 4.2 (Naive Bayes)**.**

$$y = \underset{k \in 1,\dots,K}{\operatorname{argmax}}\, p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

Where x_i is the probability of the term in the class, meaning how often it occurs in this document, divided by how often it occurs in the class. Apply add-one (laplace) smoothing such that words that don't occur in the document don't make the probability zero.

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 4.3 (Precision)**.**

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Definition 4.4 (Recall)**.**

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Definition 4.5 (F1-score)**.**

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

4.2 Quiz

Quiz 4.1 (The classification task for the 20 newsgroup text dataset is)**.**
Multi-class

Quiz 4.2 (10.000 documents, shark occurs in 10 documents)**.**
idf is $log(10000/10) = 3$

Quiz 4.3 (Document with length 100, in which the term shark occurs once)**.**
tf is 1 + log(1) = 1 (note that the document length is irrelevant for the tf)

Quiz 4.4 (Why do we need laplace smoothing for Naive Bayes?)**.**
To prevent zero probabilities for classes

5 Example Data

In supervised learning, we need data that is labelled to train our models. You can either use existing labelled data, or create new labelled data.

5.1 Existing Data

We can use a benchmark dataset (i.e. from Kaggle and the sorts), existing manually / human labelled data or labelled user-generated content.

5.1.1 Benchmark Data

This uses labelled datasets that are made specifically to evaluate and compare different methods and labels. Generally it doesn't matter that much if the data is older for developing new methods, as long as it is well-labelled and clean. Examples for text classification is the Reuters Corpus Volume 1, for named entity recognition CoNLL-2003 which uses labelled newspaper texts, for sentiment classification, IMDb movie reviews contains 50000 movie reviews that are positive or negative.

Advantages to using benchmark data is that it is high quality, reusable, available and you can compare results to other methods and approaches. However, they are not available for every specific problem and data type, and might not be usable for your task.

Another source for benchmarking datasets is Huggingface, however there is not strong curation for the datasets on Huggingface.

5.1.2 Existing Labels

Sources like papers, patents and other curated libraries have texts that are annotated with labels by humans already. For example, patents have specific classifications and categories assigned to them that show that it is relevant to a given set of labels.

Advantages are that it is high-quality data, potentially large and often freely available. However, not available for every specific problem or data type and might not be directly suitable for training classifiers.

5.1.3 Social Media (user-generated) content

Social media posts can contain tags and metadata that label its content. For example, twitter has hashtags that can be used to detect sentiment or a relevant topic to a post. Another example is customer reviews to learn sentiment and opinion, as the rating generally aligns with the sentiment of the text.

Advantages are that it is available in many languages and human-created. However, it can be inconsistent, might be low quality and is an indirect signal (it was not intended as a specific label).

5.2 Create labelled data

Sometimes, we don't have pre-existing labelled datasets that we can use for training. In this case, we first take a sample of the items (or documents) that we want to label. Second, we define a set of categories that we want to label the data with. Third, we write annotation guidelines to label the items with the categories. Finally, test with annotators whether the instructions are clear and refine the guidelines until they are. Ensure that the guidelines are clearly defined but not too trivial.

With the instructions, you can use crowdsourcing (mTurk, Crowdflower) or domain experts to annotate the texts. Compare the labels by different annotators on the same text to estimate the reliability of the data (inter-rater agreement).

Ensure that you have at at least dozens/hundreds per category, the more the better and the more difficult the problem, the more examples you need. If you use crowdsourcing, ensure that you have a check in the task, i.e. dummy questions, or say the work is compared to expert annotations.

5.2.1 Quality Control

Ensure that the same text is labelled by multiple persons, such that you can measure the agreement over labels between different annotators. This gives the reliability of the annotated data and can also be used as a measure for the difficulty of the task. A measure to use for that is Cohen's Kappa:

Definition 5.1 (Cohen's Kappa)**.**

$$\kappa = \frac{p(a) - p(e)}{1 - p(e)}$$

Where $p(a)$ is the agreed percentage, and $pr(e)$ the expected agreement based on the occurrence of each of the values.

5.3 Quiz

Quiz 5.1 (Why should we have multiple human annotators?)**.**

- *Because we need to estimate the reliability of the data.*
- *Because we need to measure the inter-rated agreement between annotators*
- *Because there is human interpretation involved in annotation*

Quiz 5.2 (What is the interpretation of Kappa = 0)**.**
Measured agreement is equal to random agreement, negative means that there is more disagreement than random.

6 Neural Models for Sequential Data

When we talk about predicting the next word in a sequence, in a classical sense we generally look at n-gram language models. Here, given a sequence of tokens, we estimate the probability distribution over the vocabulary for the next token.

In neural networks, we can use the word embeddings of the previous words to predict the next word. This works better to generalize "unseen" data. For example, in the sentence "I have to make sure that the cat gets fed.", we want to predict: "I forgot to make sure that the dog gets ...".

When we would use n-grams, we never saw "gets fed" after the word dog, but we have seen the word cat which is semantically and syntactically similar to cat and likely has a near embedding.

6.1 Sequential Text

Language is sequential data, and feed-forward neural networks do not model such temporal aspects and models the words independently from eachother. Extension of the feed-forward neural network for modelling sequential data is a recurrent neural network (RNN). This adds the weights calculated in the hidden layer in the previous time stamp. This ensures that the past context is used in calculating the output for the current input.

An example where we want to use these temporal neural networks is part-of-speech (POS) tagging, which labels (sequence labelling) words with their type (noun, verb). For example, the word fish is a noun in *the fish swims* but a verb in *we like to fish*.

A problem with using RNNs is that they only carry one time step forward in the weights. Although all previous steps are incorporated in the product of the previous weights, the relationship between words more distante is not modeled.

6.1.1 Long Short-Term Memory (LSTM)

A solution to this problem is a more complex RNN that takes a longer context into account. They have a separate vector retaining the context information that is relevant for a longer part of the sequence. This is updated in each step and carries temporally relevant data.

A problem with this is that they are slow to train, as their computation can not be parallelized. Furthermore, even though they have a longer context, they can not model relations that span multiple sentences or paragraphs.

6.2 Transformer Models

A solution to these memory problems and word embeddings are transformer models. This is basically a successor to the RNN that was generally used for the encoding and decoding of sequences but does not work with contexts well.

A transformer is a neural network with an added mechanism of "self-attention". The intuition is that while embeddings of words are learned, attention is paid to surrounding tokens and this information is integrated. In every representation that you get, it has some information about the surrounding tokens, and compares the relations of the token to all other tokens in the set (in parallel).

This makes transformer models faster than BiLSTMs and other RNNs to train. It can run in parallel because the tokens and its relations can be modeled independently.

The transformer model has an encoder-decoder architecture. In this context this means that it consists of an input encoder, transformer blocks and a language modeling head.

6.2.1 Input Encoding

The input encoder processes input tokens into a vector representation for each token, and includes with that the position of the input token in the sequence. The output of this encoder is information on what the word syntactically and semantically means and where it exists in the context.

6.2.2 Transformer blocks

Then, these input embeddings are fed into a multilayer network that also includes all previous embeddings in the sequence. It results in a set of vectors H_i that are embeddings that include the learned context of the word. These output vectors can be used as the basis for other learning tasks or for generating token sequences.

Each input embedding is compared to all other input embeddings using the dot product to calculate a score. The larger the value, the more similar it is to the vectors that are being compared. This is computationally heavy and therefore the input for transformers is maximized to a given number.

In summary, each input embedding plays three distinct roles in the self-attention mechanism:

- **The query:** Represents the current input and is used to compare against all other inputs to evaluate their relevance.
- **The key:** Represents the role or importance of each previous input in relation to the current input.
- **The value:** Provides the actual contextual information, weighted by the similarity score computed between the query and the key.

In essence, the query and key determine how much attention each input should pay to others, while the value carries the actual content being passed along. Each token in the input sequence is compared to all other tokens, and these comparisons, represented as a weighted sum, capture the contextual importance of other tokens relative to the current token.

In summary, each input embedding plays three roles in the self-attention mechanism:

6.2.3 Language modeling head

Finally, the trained / learned embeddings are passed through a final transformer block and through softmax over the vocabulary to generate a single (predicted) token.

6.3 Applying transformer models

Given a training corpus of text, we can train the transformer model to predict the next token in a sequence. The goal here is to learn the representations of meaning for words.

Then, using autoregressive generation, we incrementally generate words by repeatedly sampling the next word based on the previous words. Finally, teacher forcing forces the system to use the target tokens from training as the next input x_{t+1} instead of the decoder output y_t.

6.4 BERT

Using the transformer model as a base, we use pre-training to model a language. But, instead of going only from left-to-right, we use both sides as context when predicting words. Finally, instead of using decoding to generate the text, we stick to only encoding and use only the output embeddings. We can then use the output embeddings to do supervised learning.

This allows us to do masked language modelling. In this case, we randomly mask words during training and try to predict what these words should be. A special token is used as a boundary to separate sentences and allows to learn the relation between full sentences.

6.4.1 Fine-tuning

The output of an initial train from BERT can be used to fine-tune the model which is less computationally expensive. We take the network learned by the pretrained model and add a neural net classifier on top of it with supervised data, to perform a specific downstream task (such as named entity recognition). This basically replaces the head in the transformer model with a task-specific model. Using a pre-trained model to fine-tune is also called transfer learning.

Definition 6.1 (Transfer Learning)**.** Using a pre-trained model (such as BERT) and fine-tune the parameters using labeled data from downstream tasks (supervised learning).

For classification tasks, the input of each text is given a special token `CLS`, the output vector in the final layer for the `CLS` input represents the entire input sequence and can be used as input to a classifier head.

For sequence-pair classification, or to find the relation between two sentences, a second special token `SEP` is used to separate the two input sequences. The model processes both sequences jointly, with the `CLS` token capturing the combined representation of the pair, which is then used by the classification head to determine the relationship.

Finally, for named entity recognition, we model the head to give the label for each input token. Each token's output representation from the final layer is passed through a classification layer to predict its corresponding entity label, enabling token-level classification.

If we use a pre-trained model without fine-tuning, this is called zero-shot. This can also be used for models that were fine-tuned by someone else on a different task such as using sentence similarity for ontology mapping, newspaper benchmark on tweets or a different language.

Definition 6.2 (Zero-shot learning)**.** Using a pre-trained model without fine-tuning, or a previously fine-tuned model (on a different task).

Definition 6.3 (Few-shot learning)**.** Fine-tuning a pre-trained model on a small sample size.

Although this works very well, the due to the complexity of the transformer model it is difficult to explain why the outputs from the model are what they are. Sometimes, word models are still used to better trace back why a model does certain predictions.

6.5 Quiz

Quiz 6.1 (What is the kind of task used to learn word embeddings)**.**
Language modelling

Quiz 6.2 (Which statements about context in sequential models are true?)**.**
LSTMs are sequential models with longer memories than traditional RNNs.
BERT models compute the relation between each pair of tokens in the input.
The attention mechanism in Transformer models has quadratic complexity relative to the input length.
The maximum input length for BERT models is limited by computational memory.

Quiz 6.3 (What is the meaning of teacher forcing)**.**
Using true tokens instead of predicted tokens in generative training.

Quiz 6.4 (Consider a sentiment analysis task, what do we need to build a prediction model using transfer learning?)**.**

GPU computing, a pre-trained Chinese BERT model, a regression layer, and the 1000 items for supervised fine-tuning.

7 Generative LLMs

Generative pre-trained transformers are decoder-only transformers. Given a prompt, they can generate output text. When these transformers becomer larger in the number of parameters, they are trained for large amounts of data and fine-tuned for conversational use.

The first model that was sufficiently large to be practically useful beyond language modelling was GPT-3. If we use a pre-trained or previously fine-tuned model without fine-tuning, that is called zero-shot use.

The idea that the models get new capabilities that are not present in smaller models is called emergent abilities. These can not be predicted simply by extrapolating the performance of smaller models.

In few-shot learning, we can give a few (3-5) examples and fine-tune the model with those examples. In LLMs, these examples are given in the prompt and the actual model is not changed.

In GPT models, we sample a word in the output from the transformer's softmax distribution that results with the prompt as the left context. We then use the word embeddings for the sampled word as additional left context, and sample the next word in the same fashion. We continue generation until we reach an end-of-message marker, or a fixed length limit is reached.

Because we sample from a probability distribution, the output is probablistic and not deterministic in nature. We can do this sampling in different ways. Greedy decoding samples the most probable token at each step, which might be locally optimal but not globally.

Definition 7.1 (Greedy decoding)**.** Choose the most probably token from the softmax distribution at each step.

If we choose globally, we can take the highest possible outcome from different choices at each token. This is computationally heavy as we have to multiply all the possible choices with the successive choices.

As an alternative, we can sample from the top-k most probable tokens. When $k = 1$ the sampling is identical to greedy decoding. If we set $k > 1$ it will sometimes select a word that is probably enough, generating a more diverse text. We can also use top-p sampling, which samples from words above a cut-off probability, and dynamically increases and decreases the pool of word candidates.

Definition 7.2 (Top-k sampling)**.** Sample a word from the top k most probable words.

Definition 7.3 (Top-p sampling)**.** Sample a word from words with a probability above p.

Finally, we can use temperature based sampling, which changes the probability distribution smoothing the probability based on for example smoothing in low temperatures. A higher temperature will result in more randomness and diversity, a lower temperature produces a more focused and deterministic output.

Definition 7.4 (Temperature sampling)**.** Reshape the probability distribution instead of truncating it.

7.1 Pre-training LLMs

The pre-training task's objective in LLMs is to get as close as possible to predict the next word, using cross-entropy as a loss function. It measures the difference between a predicted probability distribution for the next word compared to the true probability distribution:

Definition 7.5 (Cross-entropy loss)**.**

$$L_{\text{CE}} = -\sum_{w \in V} y_t[w] \log(\hat{y}_t)[w]$$

Considering there is only one correct next word, it is one-hot, and the formula can be adapted:

$$L_{\text{CE}}(\hat{y}_t, y_t) = -\log(\hat{y}_t)[w_{t+1}]$$

7.1.1 Pre-training Data

Uses **Common Crawl** web data, a snapshot of the entire crawled web. Wikipedia and books. Data is filtered for quality, for example sites with PII or adult content. Boilerplate text is removed and the data is deduplicated. Finally, data is also filtered for safety, such as through the detection of illegal and toxic texts.

7.2 Evaluating Generative Models

If we want to evaluate generative models, we can generally compare it for tasks such as summarization or question answerings to human output. We can do this by measuring word overlap or semantic overlap.

It is also convenient to have a quantitative measure for the quality of a pretrained model that is not task-specific. We can measure how well the model predicts unseen text, i.e. by feeding it a text that it has not seen before and see whether it completes the text correctly.

This is also called the perplexity of the model on an unseen test set. In essence, this measures how suprised the model is to see the text. It is defined as the inverse probability that the model assigns to the test set, defined for model θ:

Definition 7.6 (Perplexity)**.**

$$\text{Perplexity}_\theta(w_{1:n}) = P_\theta(w_{1:n})^{-\frac{1}{n}}$$

7.3 Finetuning LLMs

When we fine-tune BERT models, we update the network to the supervised task by applying transfer learning. For GPT-2 this was also possible as it did not have too much parameters to do this feasibly (BERT around 110M, GPT-2 around 137M). Fine-tuning LLMs in this way is computationally not feasible, given GPT-3 has around 175B params and LLaMA3 8B or 70B params.

Instead, we can do the following:

7.3.1 Continued pretraining

We take the parameters and retrain the model on new data, using the same method of word prediction and the loss function as was done for retraining.

7.3.2 Parameter-efficient finetuning (PEFT)

We only (re-)train a subset of the parameters on new data. An example of this is LoRa: Low-Rank (dimensionality reduction) adaptation of Large Language Models. In this case, the pretrained model weights are frozen, and we inject trainable rank decomposition matrices into each layer of the transformer. This can reduce the number of trainable parameters by a 10000 times. This results into somewhat of a proxy model of the original model.

7.3.3 Supervised finetuning (SFT)

We take a small LLM with around 2 or 3 billion paramters and train it to produce exactly the sequence of tokens in the desired output.

7.3.4 Reinforcement learning from human feedback (RLHF)

Let humans indicate which output they prefer, then train the model on this distinction. This is used often in tasks where it is difficult to define a ground truth, but where humans can easily judge the quality of the generated output.

In the first step we learn a reward model from the pairwise comparisons done by humans. In the second step, we fine-tune the language model to the learned reward model, which aligns the model with human preferences.

7.4 Conversational LLMs

Conversational LLMs are GPT models that are fine-tuned for conversational usage. This is done through supervised finetuning, by training it with conversational data on the web (i.e. from Reddit). Then, RLHF was used to fine-tune the model to give appropriate and desired responses.