Natural Language Processing

Last update: 22 October 2024

Academic Year 2024 – 2025

Alma Mater Studiorum · University of Bologna

Contents

1 Basic text processing

Text normalization Operations such as:

Tokenization Split a sentence in tokens. Tokenization

| **Remark.** Depending on the approach, a token is not always a word.

Lemmatization/stemming Convert words to their canonical form. Lemmatization/stemming

| **Example.** $\{\texttt{sang}, \texttt{sung}, \texttt{sings}\} \mapsto \texttt{sing}$

Sentence segmentation Split a text in sentences. Sentence segmentation

| **Remark.** A period does not always signal the end of a sentence.

1.1 Regular expressions

Regular expression (regex) Formal language to describe string patterns. Regular expression (regex)

1.1.1 Basic operators

Disjunction (brackets) Match a single character between square brackets `[]`.

| **Example.** `/[wW]oodchuck/` matches `Woodchuck` and `woodchuck`.

Range Match a single character from a range of characters or digits.

| **Example.**
- `/[A-Z]/` matches a single upper case letter.
- `/[a-z]/` matches a single lower case letter.
- `/[0-9]/` matches a single digit.

Negation Match the negation of a pattern.

| **Example.** `/[^A-Z]/` matches a single character that is not an upper case letter.

Disjunction (pipe) Disjunction of regular expressions separated by `|`.

| **Example.** `/groundhog|woodchuck/` matches `groundhog` and `woodchuck`.

Wildcards

Optional A character followed by `?` can be matched optionally.

| **Example.** `/woodchucks?/` matches `woodchuck` and `woodchucks`.

Any `.` matches any character.

Kleene `*` A character followed by `*` can be matched zero or more times.

Kleene `+` A character followed by `+` must be matched at least once.

Counting A character followed by `{n,m}` must be matched from n to m times.

Example.

- `{n}` matches exactly n instances of the previous character.
- `{n,m}` matches from n to m instances of the previous character.
- `{n,}` matches at least n instances of the previous character.
- `{,m}` matches at most m instances of the previous character.

Anchors

Start of line `^` matches only at the start of line.

Example. `/^a/` matches a but not `ba`.

End of line `$` matches only at the end of line.

Example. `/a$/` matches a but not `ab`.

Word boundary `\b` matches a word boundary character.

Word non-boundary `\B` matches a word non-boundary character.

Aliases

- `\d` matches a single digit (same as `[0-9]`).
- `\D` matches a single non-digit (same as `[^\d]`).
- `\w` matches a single alphanumeric or underscore character (same as `[a-zA-Z0-9_]`).
- `\w` matches a single non-alphanumeric and non-underscore character (same as `[^\w]`).
- `\s` matches a single whitespace (space or tab).
- `\S` matches a single non-whitespace.

Capture group Operator to refer to previously matched substrings.

Example. In the regex `/the (.*)er they were, the \1er they will be/`, `\1` should match the same content matched by `(.*)`.

1.2 Tokenization

Lemma Words with the same stem and roughly the same semantic meaning. Lemma

Example. `cat` and `cats` are the same lemma.

Wordform Orthographic appearance of a word. Wordform

Example. `cat` and `cats` do not have the same wordform.

Vocabulary Collection of text elements, each indexed by an integer. Vocabulary

Remark. To reduce the size of a vocabulary, words can be reduced to lemmas.

Type / Wordtype Element of a vocabulary (i.e., wordforms in the vocabulary). Type / Wordtype

Token Instance of a type in a text. Token

Genre Topic of a text corpus (e.g., short social media comments, books, Wikipedia pages, ...). Genre

Remark (Herdan's law)**.** Given a corpus with N tokens, a vocabulary V over that corpus roughly have size:

$$|V| = kN^{\beta}$$

where the typical values are $10 \leq k \leq 100$ and $0.4 \leq \beta \leq 0.6$.

Stopwords Frequent words that can be dropped. Stopwords

> **Remark.** If semantics is important, stopwords should be kept. LLMs keep stopwords.

Rule-based tokenization Hand-defined rules for tokenization. Rule-based tokenization

> **Remark.** For speed, simple tokenizers use regex.

Data-driven tokenization Determine frequent tokens from a large text corpus. Data-driven tokenization

1.2.1 Data-driven tokenization

Tokenization is done by two components:

Token learner Learns a vocabulary from a given corpus (i.e., training). Token learner

Token segmenter Segments a given input into tokens based on a vocabulary (i.e., inference). Token segmenter

Byte-pair encoding (BPE) Based on the most frequent n-grams. Byte-pair encoding (BPE)

Token learner Given a training corpus C, BPE determines the vocabulary as follows:

1. Start with a vocabulary V containing all the 1-grams of C and an empty set of merge rules M.
2. While the desired size of the vocabulary has not been reached:
   a) Determine the pair of tokens $t_1 \in V$ and $t_2 \in V$ such that, among all the possible pairs, the n-gram $t_1 + t_2 = t_1 t_2$ obtained by merging them is the most frequent in the corpus C.
   b) Add $t_1 t_2$ to V and the merge rule $t_1 + t_2$ to M.

> **Example.** Given the following corpus:
>
> | Occurrences | Tokens |
> |---|---|
> | 5 | `l o w $` |
> | 2 | `l o w e r $` |
> | 6 | `n e w e s t $` |
> | 6 | `w i d e s t $` |
>
> The initial vocabulary is: $V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}\}$.
> At the first iteration, $\texttt{e} + \texttt{s} = \texttt{es}$ is the most frequent n-gram. Corpus and vocabulary are updated as:

Occurrences	Tokens
5	`l o w $`
2	`l o w e r $`
6	`n e w es t $`
6	`w i d es t $`

$$V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}\} \cup \{\texttt{es}\}$$

At the second iteration, $\texttt{es} + \texttt{t} = \texttt{est}$ is the most frequent n-gram:

Occurrences	Tokens
5	`l o w $`
2	`l o w e r $`
6	`n e w est $`
6	`w i d est $`

$$V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}, \texttt{es}\} \cup \{\texttt{est}\}$$

And so on...

Token segmenter Given the vocabulary V and the merge rules M, the BPE segmenter does the following:

1. Split the input into 1-grams.
2. Iteratively scan the input and do the following:
   a) Apply a merge rule if possible.
   b) If no merge rules can be applied, lookup the (sub)word in the vocabulary. Tokens out-of-vocabulary are marked with a special unknown token `[UNK]`.

WordPiece Similar to BPE with the addition of merge rules ranking and a special leading/tailing set of characters (usually `##`) to identify subwords (e.g., `new##`, `##est` are possible tokens). WordPiece

Unigram Starts with a big vocabulary and remove tokens following a loss function. Unigram

1.3 Normalization

Normalization Convert tokens into a standard form. Normalization

Example. `U.S.A.` and `USA` should be encoded using the same index.

Case folding Map every token to upper/lower case. Case folding

Remark. Depending on the task, casing might be important (e.g., `US` vs `us`).

Lemmatization Reduce inflections and variant forms to their base form. Lemmatization

Example. $\{\texttt{am}, \texttt{are}, \texttt{is}\} \mapsto \texttt{be}$

Remark. Accurate lemmatization requires complete morphological parsing.

Stemming Reduce terms to their stem. Stemming

Remark. Stemming is a simpler approach to lemmatization.

Porter stemmer Simple stemmer based on cascading rewrite rules.

> **Example.** `ational` $\mapsto$ `ate`, `ing` $\mapsto \varepsilon$, `sses` $\mapsto$ `ss`.

1.4 Edit distance

Minimum edit distance Minimum number of edit operations (insertions, deletions, and substitutions) needed to transform a string into another one.

Minimum edit distance

> **Remark.** Dynamic programming can be used to efficiently determine the minimum edit distance.

Levenshtein distance Edit distance where:

Levenshtein distance

- Insertions cost 1;
- Deletions cost 1;
- Substitutions cost 2.

> **Example.** The Levenshtein distance between `intention` and `execution` is 8.

```
I  N  T  E  *  N  T  I  O  N
|  |  |  |  |  |  |  |  |  |
*  E  X  E  C  U  T  I  O  N
−  ±  ±     +  ±
1  2  2     1  2
```

2 Language models

2.1 Spelling correction

Spelling correction

Spelling correction Spelling errors can be of two types:

Non-word spelling Typos that result in non-existing words. Possible candidates can be determined though a dictionary lookup.

Real-word spelling Can be:

Typographical error Typos that result in existing words.

Cognitive error Due to words similarity (e.g., `piece` vs `peace`).

Noisy channel model

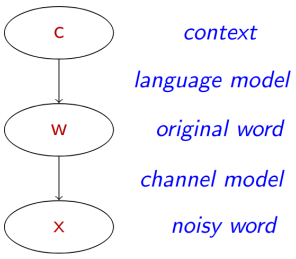
Noisy channel model Assumes that the observable input is a distorted form of the original word. A decoder tests word hypotheses and selects the best match.

More formally, we want a model of the channel that, similarly to Bayesian inference, determines the likelihood that a word $w \in V$ is the original word for a noisy one x. From there, we can estimate the correct word $\hat{w}$:

$$\hat{w} = \arg\max_{w \in V} \mathcal{P}(w|x)$$

By applying (i) Bayes' rule, (ii) the fact that $\hat{w}$ is independent of $\mathcal{P}(x)$, and (iii) that a subset $C \subseteq V$ of the vocabulary can be used, the estimate becomes:

$$\hat{w} = \arg\max_{w \in C} \underbrace{\mathcal{P}(x|w)}_{\text{channel model}} \underbrace{\mathcal{P}(w)}_{\text{prior}}$$



Moreover, it is reasonable to include a context c when computing the prior:

$$\hat{w} = \arg\max_{w \in C} \underbrace{\mathcal{P}(x|w)}_{\text{channel model}} \underbrace{\mathcal{P}(w|c)}_{\text{language model}}$$

Noisy channel spelling method Spelling correction in a noisy channel model can be done as follows:

1. Find candidate words with similar spelling to the input based on a distance metric (e.g., Damerau-Levenshtein which is the Levenshtein distance with the addition of adjacent transpositions).
2. Score each candidate based on the language and channel model:
   - Use typing features of the user.
   - Use local context.
   - Use a confusion matrix with common mistakes.

Example. Consider the sentence:

> [...] *was called a "stellar and versatile* `acress` *whose combination of sass and glamour has defined her* [...]*"*

By using the Corpus of Contemporary English (COCA), we can determine the following words as candidates:

$$\texttt{actress} \cdot \texttt{cress} \cdot \texttt{caress} \cdot \texttt{access} \cdot \texttt{across} \cdot \texttt{acres}$$

Language model By considering a language model without context, the priors are computed as $\mathcal{P}(w) = \frac{\texttt{count}(w)}{|\texttt{COCA}|}$ (where $|\texttt{COCA}| = 404\,253\,213$):

w	$\texttt{count}(w)$	$\mathcal{P}(w)$
actress	9321	0.0000231
cress	220	0.000000544
caress	686	0.00000170
access	37 038	0.0000916
across	120 844	0.000299
acres	12 874	0.0000318

Channel model By using a confusion matrix of common typos, the channel model is:

w	$x\|w$	$\mathcal{P}(x\|w)$
actress	c|ct	0.000117
cress	a|#	0.00000144
caress	ac|ca	0.00000164
access	r|c	0.000000209
across	e|o	0.0000093
acres	es|e	0.0000321
acres	ss|s	0.0000342

The ranking is the obtained as:

w	$\mathcal{P}(x\|w)\mathcal{P}(w)$
actress	$2.7 \cdot 10^9$
cress	$0.00078 \cdot 10^9$
caress	$0.0028 \cdot 10^9$
access	$0.019 \cdot 10^9$
across	$2.8 \cdot 10^9$
acres	$1.02 \cdot 10^9$
acres	$1.09 \cdot 10^9$

Therefore, the most likely correction of `acress` for this model is `across`.
If the previous word is considered in the context, the relevant tokens of the new language model are:

w_{i-1}	w_i	$\mathcal{P}(w_i\|w_{i-1})$
versatile	actress	0.000021
versatile	across	0.000021
actress	whose	0.001
across	whose	0.000006

This allows to measure the likelihood of a sentence as:

$$\mathcal{P}(\texttt{versatile actress whose}) = \mathcal{P}(\texttt{actress}|\texttt{versatile})\mathcal{P}(\texttt{whose}|\texttt{actress}) = 210 \cdot 10^{-10}$$
$$\mathcal{P}(\texttt{versatile across whose}) = \mathcal{P}(\texttt{across}|\texttt{versatile})\mathcal{P}(\texttt{whose}|\texttt{across}) = 1 \cdot 10^{-10}$$

Finally, we have that:

$$\mathcal{P}(\texttt{versatile actress whose}|\texttt{versatile acress whose}) = 2.7 \cdot 210 \cdot 10^{-19}$$
$$\mathcal{P}(\texttt{versatile across whose}|\texttt{versatile acress whose}) = 2.8 \cdot 10^{-19}$$

So `actress` is the most likely correction for `acress` in this model.

Remark. In practice, log-probabilities are used to avoid underflows and to make computation faster (i.e., sums instead of products).

2.2 Language models

(Probabilistic) language model Model to determine the probability of a word w in a given context c: Language model

$$\mathcal{P}(w|c)$$

Usually, it is based on counting statistics and uses as context the sequence of previous tokens:

$$\mathcal{P}(w_i|w_1, \dots, w_{i-1})$$

This is equivalent to computing the probability of the whole sentence, which expanded using the chain rule becomes:

$$\begin{aligned}\mathcal{P}(w_1, \dots, w_{i-1}w_i) &= \mathcal{P}(w_1)\mathcal{P}(w_2|w_1)\mathcal{P}(w_3|w_{1..2}) \dots \mathcal{P}(w_n|w_{1..n-1}) \\ &= \prod_{i=1}^{n} \mathcal{P}(w_i|w_{1..i-1})\end{aligned}$$

Remark. Simply counting the number of occurrences of a sentence as $\mathcal{P}(w_i|w_{1..i-1}) = w_{1..i}/w_{1..i-1}$ is not ideal as there are too many possible sentences.

Markov assumption Limit the length of the context to a window of k previous tokens: Markov assumption in language models

$$\mathcal{P}(w_i|w_{1..i-1}) \approx \mathcal{P}(w_i|w_{i-k..i-1})$$
$$\mathcal{P}(w_{1..n}) \approx \prod_{i=1}^{n} \mathcal{P}(w_i|w_{i-k..i-1})$$

Unigram model Model without context ($k = 0$):

$$\mathcal{P}(w_{1..n}) \approx \prod_i \mathcal{P}(w_i)$$

Bigram model Model with a single token context ($k = 1$):

$$\mathcal{P}(w_{1..n}) \approx \prod_i \mathcal{P}(w_i|w_{i-1})$$

N-gram model Model with a context of $k = N - 1$ tokens: N-gram model

$$\mathcal{P}(w_{1..n}) \approx \prod_i \mathcal{P}(w_i|w_{i-N+1..i-1})$$

Remark. N-gram models cannot capture long-range dependencies.

Estimating N-gram probabilities Consider the bigram case, the probability that a token w_i follows w_{i-1} can be determined by counting:

$$\mathcal{P}(w_i|w_{i-1}) = \frac{\texttt{count}(w_{i-1}w_i)}{\texttt{count}(w_{i-1})}$$

Remark. N-gram models cannot handle unknown tokens.

Remark. N-gram models capture knowledge about:

- Grammar and syntax.
- Some information about the dataset (e.g., domain, genre of corpus, cultural aspects, ...).

Generation by sampling Randomly sample tokens from the distribution of a language model. Generation by sampling

Remark. In N-gram models ($N \geq 2$), the distribution changes depending on the previously sampled tokens.

2.3 Metrics

2.3.1 Extrinsic evaluation

Extrinsic/downstream evaluation Compare the performance of different models on specific tasks. Extrinsic evaluation

Remark. Extrinsic evaluation is the best approach for comparing different models, but it is often computationally expensive.

2.3.2 Intrinsic evaluation

Intrinsic evaluation Measure the quality of a model independently of the task. Intrinsic evaluation

Perplexity (PP) Probability-based metric based on the inverse probability of a sequence (usually the test set) normalized by the number of words: Perplexity

$$\mathcal{P}(w_{1..N}) = \prod_i \mathcal{P}(w_i|w_{1..i-1})$$

$$\texttt{PP}(w_{1..N}) = \mathcal{P}(w_{1..N})^{-\frac{1}{N}} \in [1, +\infty]$$

A lower perplexity represents a generally better model.

Example. For bigram models, perplexity is computed as:

$$\mathcal{P}(w_{1..N}) \approx \prod_i \mathcal{P}(w_i|w_{i-1}) \qquad \texttt{PP}(w_{1..N}) = \sqrt[N]{\prod_i \frac{1}{\mathcal{P}(w_i|w_{i-1})}}$$

Remark (Perplexity intuition)**.** Perplexity can be seen as a measure of surprise of a language model when evaluating a sequence.

Alternatively, it can also be seen as a weighted average branching factor (i.e., average number of possible next words that follow any word, accounting for their probabilities). For instance, consider a vocabulary of digits and a training corpus where every digit appears with uniform probability 0.1. The perplexity of any sequence using a 1-gram model is:

$$\texttt{PP}(w_{1..N}) = \left(0.1^N\right)^{-\frac{1}{N}} = 10$$

Now consider a training corpus where 0 occurs 91% of the time and the other digits 1% of the time. The perplexity of the sequence `0 0 0 0 0 3 0 0 0 0` is:

$$\texttt{PP}(\texttt{0 0 0 0 0 3 0 0 0 0}) = \left(0.91^9 \cdot 0.01\right)^{-\frac{1}{10}} \approx 1.73$$

Remark. Minimizing perplexity is the same as maximizing the probability of the tokens.

Remark. Perplexity can be artificially reduced by using a smaller vocabulary. Therefore, it is only reasonable to compare perplexity of models with the same vocabulary.

Remark. Perplexity is generally a bad approximation of extrinsic metrics and only works well if the test set is representative of the training data. Therefore, it is only useful to guide experiments and the final evaluation should be done through extrinsic evaluation.

2.4 N-gram model problems

2.4.1 Overfitting

Overfitting

N-gram models become better at modeling the training corpus for increasing values of N. This risks overfitting and does not allow to obtain a generalized model.

Example. A 4-gram model is able to nearly perfectly generate sentences from Shakespeare's works.

2.4.2 Out-of-vocabulary tokens

There are two types of vocabulary systems:

Closed vocabulary

Closed vocabulary system All words that can occur are known.

Open vocabulary

Open vocabulary system Unknown words are possible. They are usually handled using a dedicated token `<UNK>` which allows to turn an open vocabulary system into a closed one:

- Use a vocabulary and model all other words as `<UNK>`.
- Model infrequent words as `<UNK>`.

Remark. The training set must contain `<UNK>` tokens to estimate its distribution as it is treated as any other token.

2.4.3 Unseen sequences

Only for n-grams that occur enough times a representative probability can be estimated. For increasing values of n, the sparsity grows causing many unseen n-grams that produce a probability of 0, with the risk of performing divisions by zero (e.g., perplexity) or zeroing probabilities (e.g., when applying the chain rule).

Laplace smoothing Adds 1 to all counts and renormalizes them. Given a vocabulary V and an N-gram model, smoothing is done as follows: Laplace smoothing

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-N+1..i-1}) = \frac{\texttt{count}(w_{i-N+1..i-1}w_i)+1}{\texttt{count}(w_{i-N+1..i-1})+|V|}$$

Alternatively, by only changing the numerator, it can be formulated using an adjusted count as:

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-N+1..i-1}) = \frac{c^*}{\texttt{count}(w_{i-N+1..i-1})}$$
$$c^* = \big(\texttt{count}(w_{i-N+1..i-1}w_i)+1\big)\frac{\texttt{count}(w_{i-N+1..i-1})}{\texttt{count}(w_{i-N+1..i-1})+|V|}$$

where $\frac{\texttt{count}(w_{i-N+1..i-1})}{\texttt{count}(w_{i-N+1..i-1})+|V|}$ is a normalization factor.

Example. For a 2-gram model, Laplace smoothing is computed as:

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-1}) = \frac{\texttt{count}(w_{i-1}w_i)+1}{\texttt{count}(w_{i-1})+|V|}$$

Or by using the adjusted count as:

$$\mathcal{P}_{\text{Laplace}}(w_i|w_{i-1}) = \frac{c^*}{\texttt{count}(w_{i-1})} \qquad c^* = \big(\texttt{count}(w_{i-1}w_i)+1\big)\frac{\texttt{count}(w_{i-1})}{\texttt{count}(w_{i-1})+|V|}$$

3 Text classification

3.1 Common tasks

Sentiment analysis/Opinion mining Detection of attitudes. It can involve detecting: Sentiment analysis/Opinion mining

- The holder of attitude (i.e., the source).
- The target of attitude (i.e., the aspect).
- The type of attitude (e.g., positive and negative).
- The text containing the attitude.

Spam detection

Language identification

Authorship attribution

Subject category classification

3.2 Classification

Classification task Given an input x and a set of possible classes $Y = \{y_1, \dots, y_M\}$, a classifier determines the class $\hat{y} \in Y$ associated to x. Classification task

Classification can be:

Rule-based Based on fixed (possibly handwritten) rules. Rule-based

| **Example.** Blacklist, whitelist, regex, ...

In-context learning Provide a decoder (i.e., generative) large language model a prompt describing the task and the possible classes. In-context learning

| **Example.** Zero-shot learning, few-shot learning, ...

Supervised machine learning Use a training set of N labeled data $\{(d_i, c_i)\}$ to fit a classifier. Supervised machine learning

An ML model can be:

Generative Informally, it learns the distribution of the data.

Discriminative Informally, it learns to exploit the features to determine the class.

3.3 Naive Bayes

Bag-of-words (BoW) Represents a document using the frequencies of its words. Bag-of-words (BoW)

Given a vocabulary V and a document d, the bag-of-words embedding of d is a vector in $\mathbb{N}^{|V|}$ where the i-th position contains the number of occurrences of the i-th token in d.

Multinomial naive Bayes classifier Generative probabilistic classifier based on the assumption that features are independent given the class.

Given a document $d = \{w_1, \ldots, w_n\}$, a naive Bayes classifier returns the class $\hat{c}$ with maximum posterior probability:

$$\begin{aligned}
\hat{c} &= \arg\max_{c \in C} \mathcal{P}(c|d) \\
&= \arg\max_{c \in C} \underbrace{\mathcal{P}(d|c)}_{\text{likelihood}} \underbrace{\mathcal{P}(c)}_{\text{prior}} \\
&= \arg\max_{c \in C} \mathcal{P}(w_1, \ldots, w_n|c)\mathcal{P}(c) \\
&= \arg\max_{c \in C} \prod_i \mathcal{P}(w_i|c)\mathcal{P}(c) \\
&= \arg\max_{c \in C} \sum_i \log \mathcal{P}(w_i|c) \log \mathcal{P}(c)
\end{aligned}$$

Given a training set D with N_c classes and a vocabulary V, $\mathcal{P}(w_i|c)$ and $\mathcal{P}(c)$ are determined during training by maximum likelihood estimation as follows:

$$\mathcal{P}(c) = \frac{N_c}{|D|} \qquad \mathcal{P}(w_i|c) = \frac{\texttt{count}(w_i, c)}{\sum_{v \in V} \texttt{count}(v, c)}$$

where $\texttt{count}(w, c)$ counts the occurrences of the word w in the training samples with class c.

Remark. Laplace smoothing is used to avoid zero probabilities.

Remark. Stop words can be removed from the training set as they are usually not relevant.

Remark. The likelihood part of the equation ($\sum_i \log \mathcal{P}(w_i|c)$) can be seen as a set of class-specific 1-gram language models.

Example. Given the following training set for sentiment analysis with two classes:

Class	Document
-	`just plain boring`
-	`entirely predictable and lacks energy`
-	`no surprises and very few laughs`
+	`very powerful`
+	`the most fun film of the summer`

We want to classify the sentence "`predictable with no fun`". Excluding stop words (i.e., `with`), we need to compute:

$$\mathcal{P}(\texttt{+}|\texttt{predictable with no fun}) = \mathcal{P}(\texttt{+})\mathcal{P}(\texttt{predictable}|\texttt{+})\mathcal{P}(\texttt{no}|\texttt{+})\mathcal{P}(\texttt{fun}|\texttt{+})$$
$$\mathcal{P}(\texttt{-}|\texttt{predictable with no fun}) = \mathcal{P}(\texttt{-})\mathcal{P}(\texttt{predictable}|\texttt{-})\mathcal{P}(\texttt{no}|\texttt{-})\mathcal{P}(\texttt{fun}|\texttt{-})$$

A vocabulary of 20 tokens can be used to represent the training samples. The required likelihoods and priors with Laplace smoothing are computed as:

$$\mathcal{P}(\texttt{+}) = \frac{2}{5} \qquad \mathcal{P}(\texttt{predictable}|\texttt{+}) = \frac{0+1}{9+20} \quad \mathcal{P}(\texttt{no}|\texttt{+}) = \frac{0+1}{9+20} \quad \mathcal{P}(\texttt{fun}|\texttt{+}) = \frac{1+1}{9+20}$$
$$\mathcal{P}(\texttt{-}) = \frac{3}{5} \qquad \mathcal{P}(\texttt{predictable}|\texttt{-}) = \frac{1+1}{14+20} \quad \mathcal{P}(\texttt{no}|\texttt{-}) = \frac{1+1}{14+20} \quad \mathcal{P}(\texttt{fun}|\texttt{-}) = \frac{0+1}{14+20}$$

3.3.1 Optimizations

Possible optimizations for naive Bayes applied to sentiment analysis are the following:

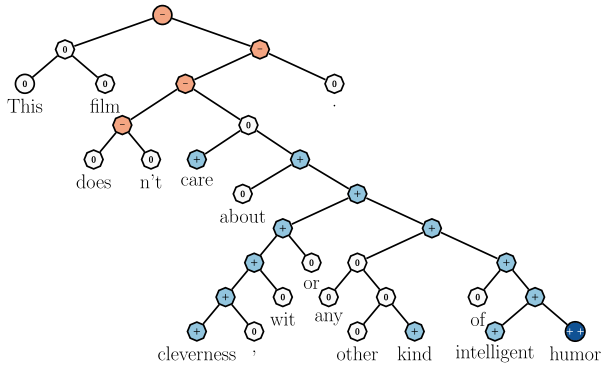
Binarization Generally, the information regarding the occurrence of a word is more important than its frequency. Therefore, instead of applying bag-of-words by counting, it is possible to produce a one-hot encoded vector to indicate which words are in the document. Binarization

Negation encoding To encode negations, two approaches can be taken: Negation encoding

Negation annotation Add to negated words an annotation so that they are treated as a new word.

> **Example.** Prepend `NOT_` to each word between a negation and the next punctuation:
>
> didn't like this movie. ↦ didn't `NOT_`like `NOT_`this `NOT_`movie.

Parse tree Build a tree to encode the sentiment and interactions of the words. By propagating the sentiments bottom-up it is possible to determine the overall sentiment of the sequence.

> **Example.** The parse tree for the sentence "`This film doesn't care about cleverness, wit or any other kind of intelligent humor.`" is the following:
>
> 
>
>
> Due to the negation (`doesn't`), the whole positive sequence is negated.

Sentiment lexicon If training data is insufficient, external domain knowledge, such as sentiment lexicon, can be used. Sentiment lexicon

> **Example.** A possible way to use a lexicon is to count the number of positive and negative words according to that corpus.

> **Remark.** Possible ways to create a lexicon are through:
>
> - Expert annotators.
> - Crowdsourcing in a two-step procedure:
>   1. Ask questions related to synonyms (e.g., which word is closest in meaning to *startle*?).
>   2. Rate the association of words with emotions (e.g., how does *startle* associate with *joy*, *fear*, *anger*, ...?).

- Semi-supervised induction of labels from a small set of annotated data (i.e., seed labels). It works by looking for words that appear together with the ones with a known sentiment.
- Supervised learning using annotated data.

3.3.2 Properties

Naive Bayes has the following properties:

- It is generally effective with short sequences and fewer data samples.
- It is robust to irrelevant features (i.e., words that appear in both negative and positive sentences) as they cancel out each other.
- It has good performance in domains with many equally important features (contrarily to decision trees).
- The independence assumption might produce overestimated predictions.

Remark. Naive Bayes is a good baseline when experimenting with text classifications.

3.4 Logistic regression

Features engineering

Features engineering Determine features by hand from the data (e.g., number of positive and negative lexicon).

Binary logistic regression

Binary logistic regression Discriminative probabilistic model that computes the joint distribution $\mathcal{P}(c|d)$ of a class c and a document d.

Given the input features $\mathbf{x} = [x_1, \dots, x_n]$, logistic regression computes the following:

$$\sigma\left(\sum_{i=1}^{n} w_i x_i\right) + b = \sigma(\mathbf{w}\mathbf{x} + b)$$

where σ is the sigmoid function.

Loss The loss function should aim to maximize the probability of predicting the correct label $\hat{y}$ given the observation $\mathbf{x}$. This can be expressed as a Bernoulli distribution:

$$\mathcal{P}(y|x) = \hat{y}^y(1-\hat{y})^{1-y} = \begin{cases} 1-\hat{y} & \text{if } y = 0 \\ \hat{y} & \text{if } y = 1 \end{cases}$$

By applying a log-transformation and inverting the sign, this corresponds to the cross-entropy loss in the binary case:

$$\mathcal{L}_{\text{BCE}}(\hat{y}, y) = -\log \mathcal{P}(y|x) = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

Optimization As cross-entropy is convex, SGD is well suited to find the parameters $\boldsymbol{\theta}$ of a logistic regressor f over batches of m examples by solving:

$$\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} \mathcal{L}_{\text{BCE}}(\hat{y}^{(i)}, f(x^{(i)}; \boldsymbol{\theta})) + \alpha \mathcal{R}(\boldsymbol{\theta})$$

where α is the regularization factor and $\mathcal{R}(\boldsymbol{\theta})$ is the regularization term. Typical regularization approaches are:

Ridge regression (L1) $\mathcal{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 = \sum_{j=1}^{n} \boldsymbol{\theta}_j^2$.

Lasso regression (L2) $\mathcal{R}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum_{j=1}^{n} |\boldsymbol{\theta}_j|$.

Multinomial logistic regression Extension of logistic regression to the multi-class case. The joint probability becomes $\mathcal{P}(y = c|x)$ and softmax is used in place of the sigmoid. Multinomial logistic regression

Cross-entropy is extended over the classes C:

$$\mathcal{L}_{\text{CE}}(\hat{y}, y) = -\sum_{c \in C} \mathbb{1}\{y = c\} \log \mathcal{P}(y = c|x)$$

3.4.1 Properties

Logistic regression has the following properties:

- It is generally effective with large documents or datasets.
- It is robust to correlated features.

Remark. Logistic regression is also a good baseline when experimenting with text classifications.
As they are lightweight to train, it is a good idea to test both naive Bayes and logistic regression to determine the best baseline for other experiments.

3.5 Metrics

3.5.1 Binary classification

Contingency table 2×2 table matching predictions to ground truths. It contains true positives (`TP`), false positives (`FP`), false negatives (`FN`), and true negatives (`TN`). Contingency table

Recall $\frac{\texttt{TP}}{\texttt{TP}+\texttt{FN}}$. Recall

Precision $\frac{\texttt{TP}}{\texttt{TP}+\texttt{FP}}$. Precision

Accuracy $\frac{\texttt{TP}+\texttt{TN}}{\texttt{TP}+\texttt{FP}+\texttt{FN}+\texttt{TN}}$. Accuracy

Remark. Accuracy is a reasonable metric only when classes are balanced.

F1 score $\frac{2 \cdot \texttt{recall} \cdot \texttt{precision}}{\texttt{recall}+\texttt{precision}}$. F1 score

3.5.2 Multi-class classification

Confusion matrix $c \times c$ table matching predictions to ground truths. Confusion matrix

Precision/Recall Precision and recall can be defined class-wise (i.e., consider a class as the positive label and the others as the negative). Precision/Recall

Micro-average precision/recall Compute the contingency table of each class and collapse them into a single table. Compute precision or recall on the pooled contingency table. Micro-average precision/recall

Remark. This approach is sensitive to the most frequent class.

Macro-average precision/recall Compute precision or recall class-wise and then average over the classes. Macro-average precision/recall

| **Remark.** This approach is reasonable if the classes are equally important.

| **Remark.** Macro-average is more common in NLP.

Example.

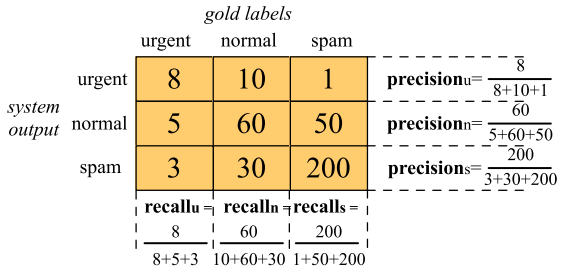


Figure 3.1: Confusion matrix

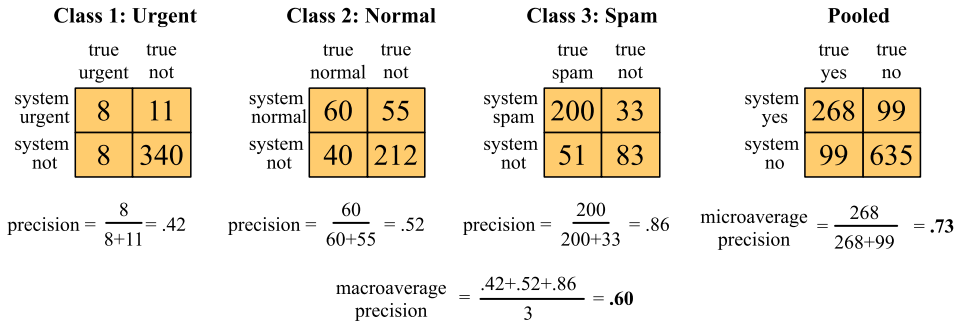


Figure 3.2: Class-wise contingency tables, pooled contingency table, and micro/macro-average precision

3.5.3 Cross-validation

n-fold cross-validation Tune a classifier on different sections of the training data: *n*-fold cross-validation

1. Randomly choose a training and validation set.
2. Train the classifier.
3. Evaluate the classifier on a held-out test set.
4. Repeat for n times.

3.5.4 Statistical significance

p-value Measure to determine whether a model A is outperforming a model B on a given test set by chance (i.e., test the null hypothesis H_0 or, in other words, test that there is no relation between A and B). *p*-value

Given:

- A test set x,
- A random variable X over the test sets (i.e., another test set),
- Two models A and B, such that A is better than B by $\delta(x)$ on the test set x,

the p-value is defined as:

$$p\text{-value}(x) = \mathcal{P}\left(\delta(X) > \delta(x) | H_0\right)$$

There are two cases:

- p-value(x) is big: the null hypothesis holds (i.e., $\mathcal{P}\left(\delta(X) > \delta(x)\right)$ is high under the assumption that A and B are not related), so A outperforms B by chance.
- p-value(x) is small (i.e., < 0.05 or < 0.01): the null hypothesis is rejected, so A actually outperforms B.

Bootstrapping test Approach to compute p-values.

Bootstrapping test

Given a test set x, multiple virtual test sets $\bar{x}^{(i)}$ are created by sampling with replacement (it is assumed that the new sets are representative). The performance difference $\delta(\cdot)$ is computed between two models and the p-value is determined as the frequency of:

$$\delta(\bar{x}^{(i)}) > 2\delta(x)$$

Remark. $\delta(x)$ is doubled due to theoretical reasons.

Example. Consider two models A and B, and a test set x with 10 samples. From x, multiple new sets (in this case of the same size) can be sampled. In the following table, each cell indicates which model correctly predicted the class:

	1	2	3	4	5	6	7	8	9	10	$A\%$	$B\%$	$\delta(\cdot)$
x	AB	A	AB	B	A	B	A	AB	–	A	0.7	0.5	0.2
$\bar{x}^{(1)}$	A	AB	A	B	B	A	B	AB	–	AB	0.6	0.6	0.0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

A possible way to sample $\bar{x}^{(1)}$ is (w.r.t. the indexes of the examples in x) $[2, 3, 3, 2, 4, 6, 2, 4, 1, 9, 1]$.

3.6 Affective meaning

The affective meaning of a text corpus can vary depending on:

Personality traits Stable behavior and personality of a person (e.g., nervous, anxious, reckless, ...).

Attitude Enduring sentiment towards objects or people (e.g., liking, loving, hating, ...).

Interpersonal stance Affective stance taken in a specific interaction (e.g., distant, cold, warm, ...).

Mood Affective state of low intensity and long duration often without an apparent cause (e.g., cheerful, gloomy, irritable, ...).

Emotion Brief response to an external or internal event of major significance (e.g., angry, sad, joyful, ...).

Remark. Emotion is the most common subject in affective computing.

3.6.1 Emotion

Theory of emotion There are two main theories of emotion:

Basic emotions Discrete and fixed range of atomic emotions. Basic emotions

> **Remark.** Emotions associated to a word might be in contrast. For instance, in the NRC Word-Emotion Association Lexicon the word `thirst` is associated to both *anticipation* and *surprise*.

Continuum emotions Describe emotions in a 2 or 3 dimensional space with the following features: Continuum emotions

Valence Pleasantness of a stimulus.

Arousal Intensity of emotion.

Dominance Degree of control on the stimulus.

> **Remark.** Valence is often used as a measure of sentiment.

4 Semantics embedding

4.1 Traditional semantic representation

Lemma/citation form Syntactic form of a word. Lemma

> **Example.** The word `pipe`.

Word sense Meaning component of a word. Word sense

Polysemous lemma Lemma with multiple senses.

> **Example.** Possible senses of the word `pipe` are: the music instrument, the conduit to transport material, ...

Supersense Semantic category for senses. Supersense

Word sense disambiguation (WSD) Task of determining the correct sense of a word. Word sense disambiguation (WSD)

4.1.1 Sense relations

Synonym Relation of (near) identity between two senses of two different words (i.e., same propositional meaning). Synonym

> **Remark** (Principle of contrast)**.** A different linguistic form is probably due to some, maybe subtle, difference in meaning.

Antonym Relation of opposition, with respect to one feature of meaning, between two senses. More specifically, antonyms can be: Antonym

- An opposition between two ends of a scale (e.g., `long`/`short`).
- A reversive (e.g., `up`/`down`).

Subordination Specificity (i.e., is-a) relation between two senses. Subordination

> **Example.** `car` is a subordinate of `vehicle`.

Superordination Generalization relation between two senses. Superordination

> **Example.** `furniture` is a superordinate of `lamp`.

Meronym Part-of relation between two senses. Meronym

> **Remark.** Relations among word senses can be seen as a graph.

4.1.2 Common ontologies

WordNet Database of semantic relations of English words. WordNet

BabelNet Multilingual database of semantic relations. BabelNet

4.1.3 Word relations

Word similarity Measure the meaning similarity of words (i.e., relation between words and not senses). Word similarity

> **Remark.** Working with words is easier than senses.

> **Example.** Cat and dog are not synonyms but have similar meaning (i.e., pets).

Word relatedness Measure the context relation of words. Word relatedness

> **Example.** `car`/`bike` are similar while `car`/`fuel` are related but not similar.

Semantic field Words that cover a particular domain and have structured relations with each other. Semantic field

> **Example.** In the context of a hospital, `surgeon`, `scalpel`, `nurse`, `anesthetic`, and `hospital` belong to the same semantic field.

Topic model Unsupervised method to cluster the topics in a document based on how a word is used in its context. Topic model

Semantic frames Words that describe the perspective or participants of a particular event. Semantic frames

> **Example.** In a commercial transaction, a `buyer` trades `money` with a `seller` in return of some `good or service`.

Semantic role labeling (SRL) Task of determining the frames and their semantic role. Semantic role labeling (SRL)

4.2 Vector semantics

Connotation Affective meaning of a word. Connotation

> **Remark.** As described in Section 3.6, emotions can be represented in a vector space. Therefore, word meanings can also be represented as vectors.

Vector semantics intuitions Vector semantics lay on two intuitions:

Distributionalism intuition The meaning of a word is defined by its environment or distribution (i.e., neighboring words). Words with a similar distribution are likely to have the same meaning. Distributionalism intuition

Vector intuition Define the meaning of a word as a point in an N-dimensional space. Vector intuition

Embedding Vector representation of a word where words with a similar meaning are nearby in the vector space. Embedding

Two common embedding models are:

TF-IDF Sparse embedding based on the counts of nearby words. TF-IDF

Word2vec Dense embedding learned by training a classifier to distinguish nearby and far-away words. Word2vec

4.2.1 Sparse embeddings

Co-occurrence matrix Matrix representing the frequency that words occur with the others. Co-occurrence matrix

Different design choices can be considered:

- Matrix design.
- Reweighing.
- Dimensionality reduction.
- Vector comparison metric.

Matrix design Shape and content of the co-occurrence matrix.

Term-document matrix Given a vocabulary V and a set of documents D, a term-document matrix has shape $|V| \times |D|$ and counts the occurrences of each word in each document. Term-document matrix

Remark. This representation allows to encode both documents (i.e., by considering the matrix column-wise) and words (i.e., by considering the matrix row-wise).

Example. An excerpt of a possible term-document matrix for Shakespeare is:

	As You Like It	*Twelfth Night*	*Julius Caesar*	*Henry V*
`battle`	1	0	7	13
`good`	114	80	62	89
`fool`	36	58	1	4
`wit`	20	15	2	3

The representation for the document *As You Like It* is $[1, 114, 36, 20]$, while the representation of the word `battle` is $[1, 0, 7, 13]$.

Word-word matrix Given a vocabulary V, a word-word matrix has shape $|V| \times |V|$. Rows represent target words and columns are context words. Given a training corpus, the word at each row is represented by counting its co-occurrences with the others within a context of N words. Word-word matrix

Remark. A larger context window captures more semantic information. A smaller window captures more syntactic information.

Example. A possible word-word matrix is:

	`aardvark`	...	`computer`	`data`	`result`	`pie`	`sugar`	...
`cherry`	0	...	2	8	9	442	25	...
`strawberry`	0	...	0	0	1	60	19	...
`digital`	0	...	1670	1683	85	5	4	...
`information`	0	...	3325	3982	378	5	13	...

Reweighing Rescale the vectors to emphasize important features and down-weigh irrelevant words.

Remark (Frequency paradox)**.** Raw frequencies are not an ideal representation for words as they are skewed and not discriminative. Moreover, overly frequent words (e.g., stop words) do not provide context information.

Term frequency-inverse document frequency (TF-IDF) Based on term-document occurrences. Given a word t and a document d, it is computed as:

Term frequency-inverse document frequency (TF-IDF)

$$\texttt{tf-idf}(t,d) = \texttt{tf}(t,d) \cdot \texttt{idf}(t)$$

where:

Term frequency (`tf`) Log-transformed frequency count of a word t in a document d:

$$\texttt{tf}(t,d) = \begin{cases} 1 + \log_{10}(\texttt{count}(t,d)) & \text{if } \texttt{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Inverse document frequency (`idf`) Inverse occurrence count of a word t across all documents:

$$\texttt{idf}(t) = \log_{10}\left(\frac{N}{\texttt{df}_t}\right)$$

where $\texttt{df}_t$ is the number of documents in which the term t occurs.

Remark. Words that occur in a few documents have a high `idf`. Therefore, stop words, which appear often, have a low `idf`.

Example. Consider the term-document matrix with `tf` in parentheses:

	As You Like It	*Twelfth Night*	*Julius Caesar*	*Henry V*
`battle`	1 (1)	0 (0)	7 (1.845)	13 (2.114)
`good`	114 (3.057)	80 (2.903)	62 (2.792)	89 (2.949)
`fool`	36 (2.553)	58 (2.763)	1 (1)	4 (1.602)
`wit`	20 (2.301)	15 (2.176)	2 (1.301)	3 (1.477)

Assume that the `df` and `idf` of the words are:

Word	`df`	`idf`
`battle`	21	0.246
`good`	37	0
`fool`	36	0.012
`wit`	34	0.037

The resulting TF-IDF weighted matrix is:

	As You Like It	*Twelfth Night*	*Julius Caesar*	*Henry V*
`battle`	0.246	0	0.454	0.520
`good`	0	0	0	0
`fool`	0.030	0.033	0.001	0.002
`wit`	0.085	0.081	0.048	0.054

Positive point-wise mutual information (PPMI) Based on term-term occurrences. Given a word w and a context word c, it determines whether they are correlated or occur by chance as follows:

Positive point-wise mutual information (PPMI)

$$\texttt{PPMI}(w,c) = \max\{\texttt{PMI}(w,c), 0\}$$

where:

Point-wise mutual information (PMI)

$$\texttt{PMI}(w, c) = \log_2 \left(\frac{\mathcal{P}(w, c)}{\mathcal{P}(w)\mathcal{P}(c)} \right) \in (-\infty, +\infty)$$

where:

- The numerator is the probability that w and c co-occur by correlation.
- The denominator is the probability that w and c co-occur by chance.

Remark. PMI > 1 indicates correlated co-occurrence. Otherwise, it is by chance.

Remark (Weighting PPMI)**.** PMI is biased towards infrequent events and returns very high values for them. This can be solved by either:

- Using add-k smoothing (typically, $k \in [0.1, 3]$).
- Slightly increasing the probability of rare context words such that $\mathcal{P}_\alpha(c) = \frac{\texttt{count}(c)^\alpha}{\sum_{c'} \texttt{count}(c')^\alpha}$ (typically, $\alpha = 0.75$).

Example. Consider the term-term matrix:

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(c)	4977	5673	473	512	61	11716

The PPMI between information and data con be computed as:

$$\begin{aligned}
\mathcal{P}(\texttt{information}, \texttt{data}) &= \frac{3982}{11716} = 0.3399 \\
\mathcal{P}(\texttt{information}) &= \frac{7703}{11716} = 0.6575 \\
\mathcal{P}(\texttt{data}) &= \frac{5673}{11716} = 0.4872 \\
\texttt{PPMI}(\texttt{information}, \texttt{data}) &= \max\left\{ \log_2 \left(\frac{0.3399}{0.6575 \cdot 0.4872} \right), 0 \right\} = 0.0944
\end{aligned}$$

Remark. Reweighing loses information about the magnitude of the counts.

Dimensionality reduction Reduce the dimensionality of the embeddings.

Vector comparison Metric to determine the distance of two embeddings.

Dot product $\mathbf{w} \cdot \mathbf{v} = \sum_{i=1}^{n} w_i v_i$.

Length Compare the length $|\mathbf{v}| = \sqrt{\sum_{i=1}^{n} v_i^2}$ of the vectors.

Cosine similarity $\frac{\mathbf{w} \cdot \mathbf{v}}{|\mathbf{w}|\,|\mathbf{v}|}$.

4.2.2 Dense embeddings

Remark. Dense embeddings are usually:

- Easier to process with machine learning algorithms.
- Able to generalize better than simply counting.
- Handle synonyms better.

Neural language modeling Use a neural network to predict the next word w_{n+1} given an input sequence $w_{1..n}$. The general flow is the following: Neural language modeling

1. Encode the input words into one-hot vectors ($\mathbb{R}^{|V| \times n}$).
2. Project the input vectors with an embedding matrix $\boldsymbol{E} \in \mathbb{R}^{d \times |V|}$ that encodes them into d-dimensional vectors.
3. Pass the embedding into the hidden layers.
4. The final layer is a probability distribution over the vocabulary ($\mathbb{R}^{|V| \times 1}$).

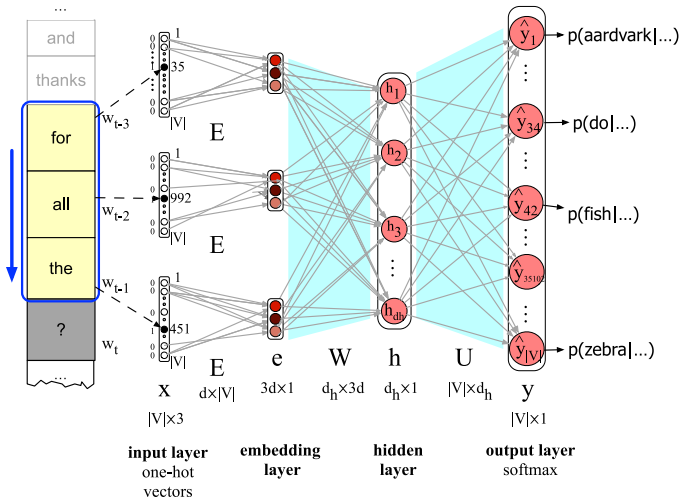


Figure 4.1: Example of neural language model with a context of 3 tokens

Remark. The embedding matrix $\boldsymbol{E}$ can be used independently to embed words. In fact, by construction, the i-th column of $\boldsymbol{E}$ represents the embedding of the i-th token of the vocabulary.

Training Given a text corpus, training is done sequentially in a self-supervised manner by sliding a context window over the sequence. At each iteration, the next word is predicted and cross-entropy is used as loss.

Remark. The initial embedding matrix is usually initialized using statistical methods and not randomly.

Word2vec Based on the idea of using a binary classifier to determine whether a word c is likely to appear near the target word w. Word2vec

Given a context word c and a target word w, the problem can be solved using a logistic regressor (i.e., use the dot product to measure vector similarity):

$$\mathcal{P}(+|w,c) = \sigma(\mathbf{c} \cdot \mathbf{w}) \qquad \mathcal{P}(-|w,c) = 1 - \mathcal{P}(+|w,c)$$

where $\mathbf{w} \in \mathbb{R}^d$ and $\mathbf{c} \in \mathbb{R}^d$ are the columns of the learned embedding matrix for the words w and c, respectively.

Moreover, it is assumed that context words are independent, therefore, if the context is a sequence, it is computed as follows::

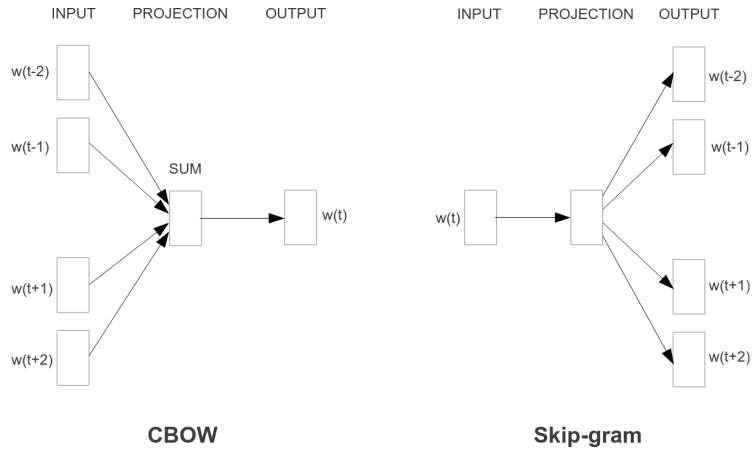
$$\mathcal{P}\left(+|w, c_{1..L}\right) = \prod_{i=1}^{L} \sigma(\mathbf{c}_i \cdot \mathbf{w})$$

Training Given a text corpus, chosen target words and their neighbors are considered positive examples. Negative examples are obtained by randomly sampling other words.

When training, two variants are possible:

Continuous bag-of-words (CBOW) Given the context words, predict the target word.

Skip-grams Given the target word, predict the (position independent) context words.



Remark. In practice, Word2vec learns two sets of embeddings $\boldsymbol{W} \in \mathbb{R}^{|V| \times d}$ and $\boldsymbol{C} \in \mathbb{R}^{|V| \times d}$ for the target and context words, respectively. At the end, they can either be averaged, concatenated, or one can be dropped.