Natural Language Processing

Fabian Bosshard

February 4, 2026

Contents

1 Useful formulae

Conditional probability: Let $(\Omega, \mathcal{A}, \mathrm{P})$ be a probability space. Recall that the conditional probability of an $A \in \mathcal{A}$ given $B \in \mathcal{A}$ is defined as

$$\mathrm{P}(A \mid B) := \begin{cases} \frac{\mathrm{P}(A \cap B)}{\mathrm{P}(B)} & \mathrm{P}(B) > 0 \\ 0 & \mathrm{P}(B) = 0 \end{cases} \tag{1.1}$$

Theorem 1.1 (Chain rule of probability)**.** For any events $A_1, \ldots, A_n$:

$$\mathrm{P}\left(\bigcap_{i=1}^{n} A_i\right) = \prod_{k=1}^{n} \mathrm{P}\left(A_k \mid \bigcap_{j=1}^{k-1} A_j\right) \tag{1.2}$$

e.g. with 4 variables: $\mathrm{P}(A, B, C, D) = \mathrm{P}(A)\ \mathrm{P}(B \mid A)\ \mathrm{P}(C \mid A, B)\ \mathrm{P}(D \mid A, B, C)$. ◁

Proof. Apply Equation 1.1 repeatedly. □

Cosine similarity between two vectors $\boldsymbol{A}$ and $\boldsymbol{B}$:

$$\text{similarity} = \cos(\angle(\boldsymbol{A}, \boldsymbol{B})) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{\|\boldsymbol{A}\|\|\boldsymbol{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1.3}$$

2 Text Classification

2.1 Text preprocessing

Prior to tokenization:

- remove markup (non-content information), e.g. `<HTML>` tags
- lowercase text to reduce vocabulary size (can lose information: e.g. "WHO" vs "who")
- remove punctuation

After tokenization:

- remove stopwords[1] (convey little information about the topic of a text)
- remove low-frequency words (since insufficient information present to determine correlation between word presence and class)

less common activities:

- stemming or lemmatization to reduce vocabulary size
- spelling correction

ASCII encoding keyboard $\to [1, 128]$

UTF-8 encoding very vast (160 languages, 149k Unicode characters)

Definition 2.1 (Morpheme)**.** Smallest linguistic unit that has semantic meaning. ◀

2.2 Bag of words

Vocabulary of document provides most important signal; # occurrences of words provides further information.

Bag-of-words model:

- represent documents as vectors of word counts
- sparse representation (long vector with many zeros)
- completely ignores word order
- extension to include n-grams can increase performance: but greatly increases number of dimensions, so more data is then needed

usually have far fewer documents than vocabulary terms, which means fewer examples than features

- means multiple settings of parameters can explain observed class labels
- strong regularization needed to prevent overfitting

2.2.1 Heaps' law

Vocabulary grows with approximately the square root of document/collection length:

$$\boxed{V(l) \propto l^{\beta}, \quad \beta \approx 0.5}$$

[1] high-frequency terms in language, e.g. "the", "is", "and"

2.2.2 Zipf's law

Define as the collection term frequency ctf_t of a token t ("token frequency") as the number of times it occurs in the entire corpus:

$$\text{ctf}_t := \sum_{d \in \mathcal{D}} \text{count}(t, d)$$

Token frequency approximately proportional to the inverse of its rank:

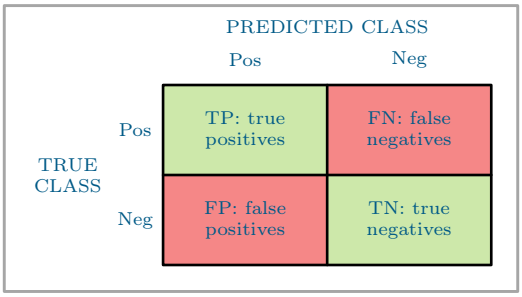
$$\boxed{\text{ctf}_t \propto \frac{1}{\text{rank}(t)^s}, \quad s \approx 1}$$

Most common word occurs approximately twice as often as the next common one, three times as often as the third most common, and so on.

2.3 Evaluating text classifiers

2.3.1 Evaluating a Binary Text Classifier

First thing to investigate is the confusion matrix:



- Accuracy = % of correct predictions:

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{TP} + \text{FN} + \text{FP}}$$

- Precision = % of positive predictions that were correct:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- Recall = % of positive instances that were found by model:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- F-measure = harmonic mean of precision and recall:

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$$

- AUC = area under the ROC curve is often used as a single measure that doesn't depend on the confidence threshold used to make predictions with the model

2.3.2 Evaluating multi-class classifiers

if there are n classes,

- confusion matrix will be $n \times n$
- by considering it positive class in a one-vs-all setting
- combine each class's precision and recall values into a single measure:
  - **macro-average**: average over classes weighting each class the same
  - **micro-average**: average over classes weighting each class by the number of data-points in it

3 Text search and clustering

3.1 tf-idf: term frequency - inverse document frequency

Definition 3.1 (inverse document frequency)**.**

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t} \tag{3.1}$$

where df_t is the number of documents in the corpus which contains t (document frequency), while N is the number of documents in total (corpus size). ◀

high idf ⇒ few documents matching; $\mathrm{idf}_t = 0 \Rightarrow t$ is in every document ($\log 1 = 0$).

Definition 3.2 (term frequency)**.**

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10}(\mathrm{count}(t,d)) & \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

◀

a term t which occur 0 times in a document d has $\mathrm{tf}_{t,d} = 0$; if it occurs once, $\mathrm{tf}_{t,d} = 1+\log_{10}(1) = 1$; if it occurs 10 times, $\mathrm{tf}_{t,d} = 1 + \log_{10}(10) = 2$; and so on.

Definition 3.3 (tf-idf)**.**

$$\text{tf-idf}(t,d) = \mathrm{tf}_{t,d} \cdot \mathrm{idf}_t \tag{3.3}$$

◀

Definition 3.4 (BM25)**.** goto method for term-based text retrieval...

$$\sum_{t \in q} \mathrm{idf}_t \; \frac{\mathrm{tf}_{t,d}}{\mathrm{tf}_{t,d} + k\left(1 - b + b\frac{|d|}{d_{\mathrm{avg}}}\right)}$$

- t terms in q query
- $k \rightarrow$ adjusts balance between term frequency and IDF
- $b \rightarrow$ controls the importance of document length normalization
- $|d|$ is length of the document; d_{avg} is the average on the document collection ◀

3.1.1 Scoring documents

Vector space model treates tf-idf values for all terms in document as a vector representation:

$$\boldsymbol{d} = \big[\text{tf-idf}(t_1, d), \text{tf-idf}(t_2, d), \ldots, \text{tf-idf}(t_n, d)\big]$$

similarity between query vector $\boldsymbol{q}$ and document vector $\boldsymbol{d}$ computed using cosine similarity:

$$\mathrm{score}(q,d) = \cos(\boldsymbol{q}, \boldsymbol{d}) = \frac{\boldsymbol{q}}{\|\boldsymbol{q}\|} \cdot \frac{\boldsymbol{d}}{\|\boldsymbol{d}\|}$$

3.2 Evaluating search engines

- precision at depth k: $P@k = (\#\text{ relevant docs in top } k)/k$
- recall at depth k: $R@k = (\#\text{ relevant docs in top } k)/(\#\text{ relevant docs in total})$
- F-measure at depth k:

$$F_1@k = \frac{1}{\big((1/P@k + 1/R@k)/2\big)}$$

- AveP = MAP: mean average precision (estimate area under precision/recall curve)

$$\mathrm{AveP} = \frac{\sum_{k=1}^{n} P@k \; \mathrm{rel}(k)}{\#\text{ relevant docs}}$$

  - bool $\mathrm{rel}(k) = 1$ if document at position k is relevant

4 Language models and word embeddings

4.1 Markov model

$$\mathrm{P}(\text{croquet} \mid \text{play}) = \frac{N(\text{play croquet})}{N(\text{play})}, \qquad \mathrm{P}(\text{croquet} \mid \text{to play}) = \frac{N(\text{to play croquet})}{N(\text{to play})}$$

Longer n-grams give better prediction but also less chance of finding the exact same sequence.

Smoothing: add a small constant to all counts before estimating probabilities, e.g. Laplace (with bigram assumption):

$$\mathrm{P}_{\text{Laplace}}(w_n \mid w_{n-1}) = \frac{\text{count}(w_{n-1}\, w_n) + 1}{\sum_w (\text{count}(w_{n-1}\, w) + 1)} = \frac{\text{count}(w_{n-1}\, w_n) + 1}{\text{count}(w_{n-1}) + V}$$

Backoff: if an n-gram is unknown, use $(n-1)$-gram

Interpolate between n-grams, e.g. trigram-bigram-unigram:

$$\mathrm{P}(w_n \mid w_{n-1}\, w_{n-2}) = \lambda_1\, \mathrm{P}(w_n \mid w_{n-1}\, w_{n-2}) + \lambda_2\, \mathrm{P}(w_n \mid w_{n-1}) + \lambda_3\, \mathrm{P}(w_n), \qquad \sum_i \lambda_i = 1$$

choose lambdas that maximize probability on held-out **development** data

4.2 Evaluating

extrinsic: use model in a downstream task (e.g. as a spelling corrector) and evaluate performance on that task

intrinsic:

- train parameters of model on training set and test model performance on held -out dataset
- use likelihood that model would produce the new data to evaluate how well it is doing

4.2.1 Perplexity

of language model quantifies level of surprise/confusion at seeing new text

- measures how unlikely the observed data is under the model

Calculating perplexity:

1. *compute probability* of observed sequence $\mathrm{P}(w_1, w_2, \ldots, w_n)$ under model
   - e.g. for a bigram m: $\mathrm{P}(w_1, \ldots, w_n) = \mathrm{P}(w_1)\, \mathrm{P}(w_2 \mid w_1)\, \mathrm{P}(w_3 \mid w_2) \cdots \mathrm{P}(w_n \mid w_{n-1})$
2. *normalize probability* for length of text sequence
   - i.e. compute "average" per-word probability: $\sqrt[n]{\mathrm{P}(w_1, w_2, \ldots, w_n)}$
3. *invert probability* to compute uncertainty:

$$\text{Perplexity}(w) = \big(1/\,\mathrm{P}(w_1, w_2, \ldots, w_n)\big)^{1/n} = \sqrt[n]{\frac{1}{\mathrm{P}(w_1, w_2, \ldots, w_n)}}$$

4.3 Embeddings

To overcome the problem of data sparsity in n-gram language models it would be good to have a similar representation for similar words.

4.3.1 WordNet

WordNet organizes lexical information in terms of word meanings (senses), rather than word forms (actual words). In that respect, WordNet resembles a thesaurus more than a dictionary.

- attempt to model human lexical memory
- is essentially an ontology (a carving up of the world's meanings)
- no longer SOTA, since difficult to build, language-specific, incomplete, still difficult to quantify similarity

4.3.2 Distributional hypothesis of Harris (1968)

words that occur in the same contexts tend to have similar meanings.

- **Practical implication**:
  We can find similar words by comparing their distribution over contexts.
- Count how often words appear in several contexts; contexts become dimensions and target words w, v become vectors $\boldsymbol{w}, \boldsymbol{v}$, then compute $\cos(\boldsymbol{w}, \boldsymbol{v})$. gives a value in $[0, 1]$ of similarity.
- Then apply dimensionality reduction (SVD / Latent Semantic Analysis): reducing millions of contexts to hundreds of dimensions, while keeping as much of the info as possible (now it's **dense** embedding).

While tf-idf vectors are very long and full of 0s (sparse), learnt vectors are shorter and with full of non-0 (dense).

4.3.3 Sparse versus dense vectors

Vocabulary, all the words your model covers

Count-based (or tf-idf) vectors are

- **long** (length $|V| = 20{,}000$ to $50{,}000$)
- **sparse** (most elements are zero)

Alternative: learn vectors which are

- **short** (length = 50 to 1000)
- **dense** (most elements are non-zero)

Why dense vectors?

- short vectors may be easier to use as *features* in machine learning (fewer weights to learn)
- may *generalize* better than explicit counts
- emprically, work better

4.4 Word2vec

Train a classifier on a binary prediction task: "Is word w likely to show up in context c?"

We don't care about the task, but we take learned classifier weights as word embeddings.

big idea: **self-supervision**

Different versions, we focus on **skip-gram** with negative sampling (SGNS).

Skip-gram: given the central word, predict context.

One context word c:

$$P(+ \mid w, c) = \sigma(\boldsymbol{c} \cdot \boldsymbol{w}) = \frac{1}{1 + \exp(-\boldsymbol{c} \cdot \boldsymbol{w})}$$

We have lots of context words. We'll assume independence and just multiply them:

$$P\left(+ \mid w, c_{1:L}\right) = \prod_{i=1}^{L} \sigma\left(\boldsymbol{c}_i \cdot \boldsymbol{w}\right)$$

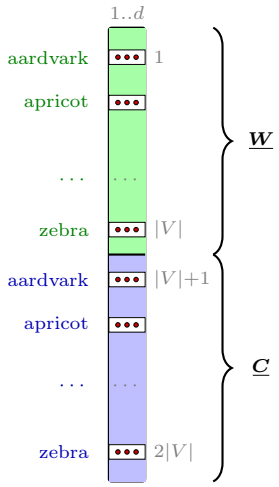
$$\log P(+ \mid w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(\boldsymbol{c}_i \cdot \boldsymbol{w})$$

Train via SGD!

Word weights will be adjusted such that

- positive pairs will be more likely
- negative pairs will be less likely

over the entire training set.



SGNS learns 2 sets of embeddings:

- target embeddings matrix $\underline{\boldsymbol{W}}$
- context embeddings matrix $\underline{\boldsymbol{C}}$

it is common to just add them together, representing word i as $\boldsymbol{w}_i + \boldsymbol{c}_i$.

Fasttext

- addresses problem of unknown words
- imagine language with a lot of morphology
- uses subword segmentation to deal with this

4.5 Properties of Word Embeddings

many interesting and somewhat surprising properties

semantic clustering:

- neighbors in embedding space are semantically related to one another

provide dense distributed representation

- where individual dimensions aren't usually interpretable

but translation in the space is meaningful

- certain directions in the space have meaning
- which means that semantics is often additive
- and analogies are encoded in the embedding: e.g. `man` is to `woman` as `king` is to `queen`

5 Sequence classification and labelling

CRF: conditional random fields.

Part of speech (POS): assigning to each word a word class (noun, verb etc), so a map from words to classes.

5.1 Named entity recognition (NER)

Similar to POS but instead of noun etc classes like "person", "organisation", etc.

NER can be hard because of:

- *segmentation*: in POS tagging each word gets one tag, while in NER entities can be phrases
- *type ambiguity*: same word/phrase can have many types depending on context

6 Sequence-to-sequence model

Soft-Attention produces weighted average over input embeddings:

$$w_{i,j} = \mathrm{P}(j \mid i) = \mathrm{softmax}\big(\mathrm{similarity}(\boldsymbol{h}_{i-1}, \boldsymbol{e}_j)\big), \qquad \boldsymbol{z}_i = \sum_j w_{i,j}\, \boldsymbol{e}_j$$

Calculating similarity:

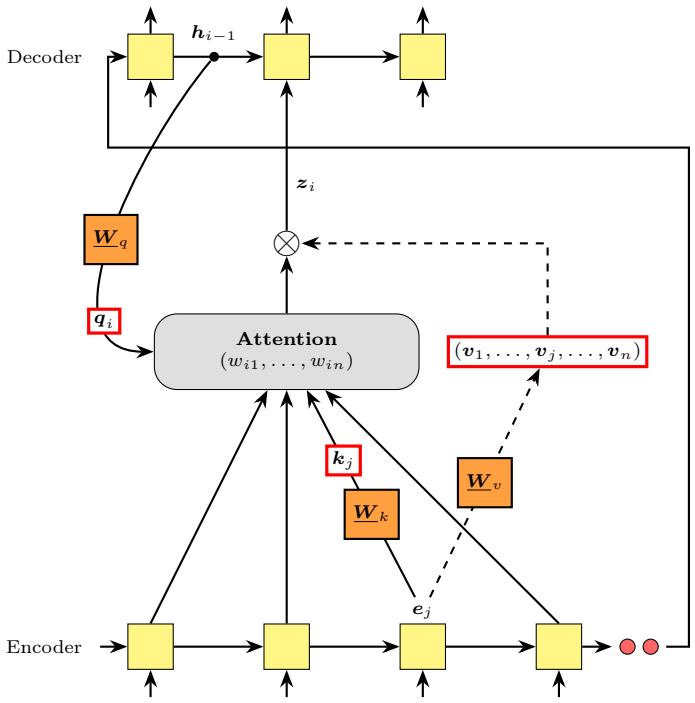
- **additive** attention: concatenate decoder state and encoder embedding in a feedforward network:
$$\mathrm{similarity}(\boldsymbol{h}_{i-1}, \boldsymbol{e}_j) = \mathrm{FFNN}(\boldsymbol{h}_{i-1} \parallel \boldsymbol{e}_j)$$
- **multiplicative** attention: dot product of encoder vector and decoder previous state,
$$\mathrm{similarity}(\boldsymbol{h}_{i-1}, \boldsymbol{e}_j) = \frac{\boldsymbol{h}_{i-1} \cdot \boldsymbol{e}_j}{\sqrt{d}}$$
where d is the embedding size.
  - we divide by $\sqrt{d}$ so that the standard deviation remains 1:
$$\text{if } a_i, b_i \sim \mathcal{N}(0,1) \Rightarrow \mathrm{Var}[a_i b_i] = 1$$
$$\Rightarrow \mathrm{Var}[\boldsymbol{a} \cdot \boldsymbol{b}] = \sum_{i=1}^{d} \mathrm{Var}[a_i b_i] = d$$

Generalizing the notation with queries (what is being looked up), keys (index of what to find), values (stored information at key):

$$\boldsymbol{z}_i = \sum_j \mathrm{softmax}\Big(\frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d}}\Big)\, \boldsymbol{v}_j$$

where the queries, keys, and values are usually obtained by a linear transformation:

$$\boldsymbol{q}_i = \underline{\boldsymbol{W}}_q \boldsymbol{h}_{i-1}, \qquad \boldsymbol{k}_j = \underline{\boldsymbol{W}}_k \boldsymbol{e}_j, \qquad \boldsymbol{v}_j = \underline{\boldsymbol{W}}_v \boldsymbol{e}_j, \qquad \underline{\boldsymbol{W}}_{q,k,v} \in \mathbb{R}^{d \times d}$$



Interpreting queries, keys, values with a hypothetical example:

- **query**: what is being looked up
need adjective that describes a person
- **key:** index of what to find
have adjective that describes people & animals
- **value**: stored information at key
word is "friendly", which in Italian could be translated to "simpatico"