Transforming tokens into features

Bag of words (BOW)

Let's count occurrences of a particular token in our text

- Motivation: we're looking for marker words like "excellent" or "disappointed"
- For each token we will have a feature column, this is called **text vectorization**.

Sparse

good movie
not a good movie
did not like

good	movie	not	a	did	like
1	**1**	0	0	0	0
1	**1**	**1**	**1**	0	0
0	0	**1**	0	**1**	**1**

- Problems:
  - we loose word order, hence the name "bag of words"
  - counters are not normalized

Let's preserve some ordering

We can count token pairs, triplets, etc.

- Also known as n-grams
  - 1-grams for tokens
  - 2-grams for token pairs
  - …

good movie
not a good movie
did not like

good movie	movie	did not	a	…
1	**1**	0	0	…
1	**1**	0	**1**	…
0	0	**1**	0	…

- Problems:
  - too many features

Remove some n-grams

Let's remove some n-grams from features based on their occurrence frequency in documents of our corpus

Remove some n-grams

Let's remove some n-grams from features based on their occurrence frequency in documents of our corpus

- **High frequency n-grams:**
  - Articles, prepositions, etc. (example: and, a, the)
  - They are called **stop-words**, they won't help us to discriminate texts → remove them
- **Low frequency n-grams:**
  - Typos, rare n-grams
  - We don't need them either, otherwise we will likely overfit
- **Medium frequency n-grams:**
  - Those are good n-grams

There're a lot of medium frequency n-grams

- It proved to be useful to look at n-gram frequency in our corpus for filtering out bad n-grams
- What if we use it for ranking of medium frequency n-grams?
- **Idea**: the n-gram with smaller frequency can be more discriminating because it can capture a specific issue in the review (Ex: wifi breaks often)

Rank the medium frequency n-grams to avoid the unnecessary words.

TF-IDF

Term frequency (TF)

- $\mathrm{tf}(t, d)$ – frequency for term (or n-gram) t in document d
- Variants:

weighting scheme	TF weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$
log normalization	$1 + \log(f_{t,d})$

https://en.wikipedia.org/wiki/Tf–idf

TF-IDF

Inverse document frequency (IDF)

- $N = |D|$ – total number of documents in corpus
- $|\{d \in D : t \in d\}|$ – number of documents where the term t appears
- $\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$

TF-IDF

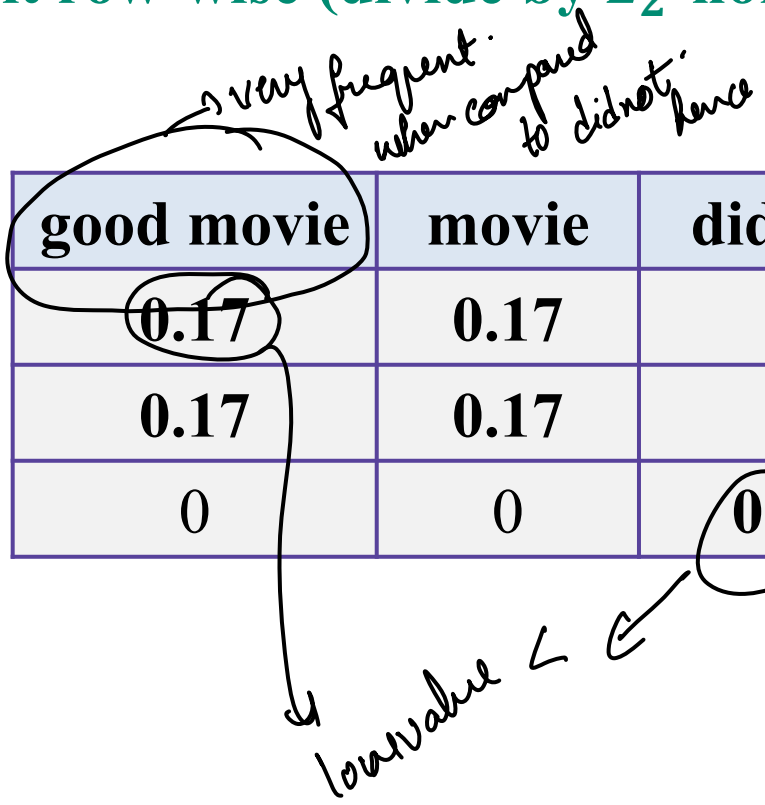
- $\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$
- A high weight in TF-IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents

Better BOW

- **Replace counters with TF-IDF**
- **Normalize the result row-wise (divide by L_2-norm)**

good movie
not a good movie
did not like

good movie	movie	did not	…
0.17	**0.17**	0	…
0.17	**0.17**	0	…
0	0	**0.47**	…



Python TF-IDF example

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
texts = [
    "good movie", "not a good movie", "did not like",
    "i like it", "good one"
]
tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1, 2))
features = tfidf.fit_transform(texts)
pd.DataFrame(
    features.todense(),
    columns=tfidf.get_feature_names()
)
```

	good movie	like	movie	not
0	0.707107	0.000000	0.707107	0.000000
1	0.577350	0.000000	0.577350	0.577350
2	0.000000	0.707107	0.000000	0.707107
3	0.000000	1.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	0.000000

Summary

- We've made simple counter features in bag of words manner
- You can add n-grams → to preserve the rank of words.
- You can replace counters with TF-IDF values
- In the next video we will train our first model on top of these features