

N-Grams

N-grams: Unigrams

- Given a corpus of text, the *n-grams* are the sequences of *n consecutive* words that are in the corpus

- example (*12 word sentence*)

the cat that sat on the sofa also sat on the mat

n=1 (8 unigrams)

- the 3 sat 2
- on 2 cat 1
- that 1 sofa 1
- also 1 mat 1

N-grams: Bigrams

example (12 word sentence)


the cat that sat on the sofa also sat on the mat

\longleftrightarrow
2 words

N=2 (8 *bigrams*)

- sat on 2 on the 2
- the cat 1 cat that 1
- that sat 1 the sofa 1
- sofa also 1 also sat 1
- the mat 1

N-grams: Trigrams

- **example (12 word sentence)**
- the cat that sat on the sofa also sat on the mat
- 
- **N=3 (9 trigrams)**
- *most language models stop here, some stop at quadrigrams*
- too many n-grams
- low frequencies
- sat on the 2 the cat that 1
- cat that sat 1 that sat on 1
- on the sofa 1 the sofa also 1
- sofa also sat 1 also sat on 1
- on the mat 1

N-grams: Quadrigrams

- Example: (12 word sentence)
- the cat that sat on the sofa also sat on the mat

← 4 words →

- N=4 (8 quadrigrams)
- | | |
|----------------------|---------------------|
| • the cat that sat 1 | cat that sat on 1 |
| • that sat on the 1 | sat on the sofa 1 |
| • on the sofa also 1 | the sofa also sat 1 |
| • sofa also sat on 1 | also sat on the 1 |
| • sat on the mat 1 | |

How Many Words?

- *I do uh main- mainly business data processing*
 - Fragments
 - Filled pauses
- Are **cat** and **cats** the same word?
- Some terminology
 - **Lemma**: a set of lexical forms having the same stem, major part of speech, and rough word sense
 - **Cat** and **cats** = same lemma
 - **Wordform**: the full inflected surface form.
 - Cat and cats = different wordforms

How Many Words?

- *they picnicked by the pool then lay back on the grass and looked at the stars*
 - 16 tokens
 - 14 types
- Brown et al (1992) large corpus
 - 583 million wordform tokens
 - 293,181 wordform types
- Google
 - Crawl 1,024,908,267,229 English tokens
 - 13,588,391 wordform types

Application- sentiment analysis

- (We are looking for marker words like “excellent” or “disappointed”. For each token we have a feature column, called **text vectorization**)

| | good | movie | not | a | did | like |
|------------------|------|-------|-----|---|-----|------|
| good movie | 1 | 1 | 0 | 0 | 0 | 0 |
| not a good movie | 1 | 1 | 1 | 1 | 0 | 0 |
| did not like | 0 | 0 | 1 | 0 | 1 | 1 |

- Problem
 - Loose word orders, hence the name bag of words (BOW)
 - Non-normalized counters
- Count token pairs, triples, etc.
- Problem
 - To many features

| | good movie | movie | did not | a | ... |
|------------------|------------|-------|---------|---|-----|
| good movie | 1 | 1 | 0 | 0 | ... |
| not a good movie | 1 | 1 | 0 | 1 | ... |
| did not like | 0 | 0 | 1 | 0 | ... |

Remove some n-grams

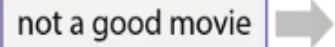
- High freq n-grams:
 - Articles, prepositions, ..(e.g. and, a, the) = stop words. Don't help us to discriminate text
- Low freq n-grams
 - Types, rare n-grams
 - Don't need them, otherwise will likely overfit
- Medium freq
 - Good
- Smaller freq is more discriminating because it can capture a specific issue in the review => tf-idf

TF-IDF

- $Tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$
- Term frequency: $Tf(t, d)$ -
 - freq for term(n-gram) t in document d
 - Variants:
- $IDF(t, D)$ -inverse doc freq = $\log \frac{N}{df}$
 - $N=|D|$ - total number of documents
 - $Df = |\{d \in D \mid t \in d\}|$ - number of docs term t appears in it
- High weights is reached by a high term frequency (in the given doc) and low document frequency of term in the whole collection of documents
- Better BOW (replace counters with tf-idf, normalize the result row-wise (devide by l2-norm))

| 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 | $1 + \log(f_{t,d})$ |

- Replace counters with tf-idf
- Normalize the result row-wise (using L2-norm)



| | good movie | movie | did not | ... |
|------------------|------------|-------|---------|-----|
| good movie | 0.17 | 0.17 | 0 | ... |
| not a good movie | 0.17 | 0.17 | 0 | ... |
| did not like | 0 | 0 | 0.47 | ... |

- **Python:**

```
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()
)
```

Imdb movie review DB

- <http://ai.stanford.edu/~amaas/data/sentiment>
- Classes: 0-4 stars→class0 7-10 stars→class1
- 50/50 test train. Evaluation: accuracy
- Feature: Bag of 1-grams with tf-idf values → sparse
- Model: logistic regression (handles sparsity, fast to train, weights can be interpreted)
- Accuracy=88.5%,
- Weights:

| ngram | weight | VS | ngram | weight |
|--------------|----------|----|--------------|------------|
| great | 9.042803 | | worst | -12.748257 |
| excellent | 8.487379 | | awful | -9.150810 |
| perfect | 6.907277 | | bad | -8.974974 |
| best | 6.440972 | | waste | -8.944854 |
| wonderful | 6.237365 | | boring | -8.340877 |
| Top positive | | | Top negative | |

improvement

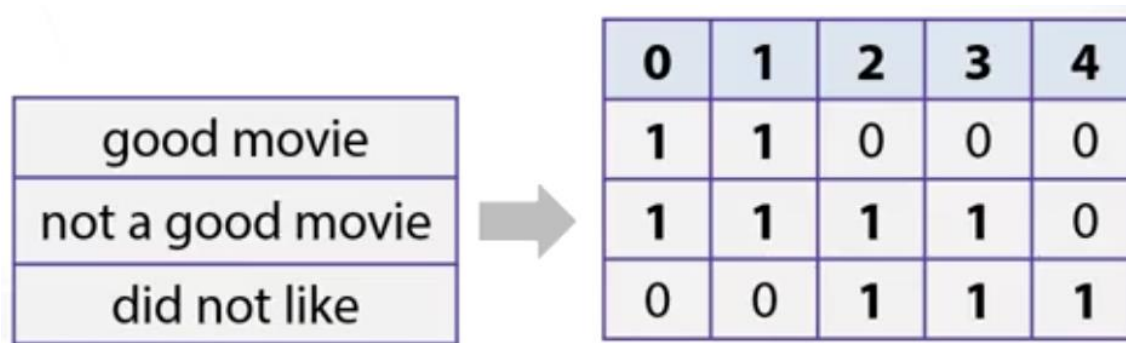
- Use n-grams
- Token normalization
- Have special tokens for emoji, :), !!!, ...
- Model with SVM, naïve bayes,...
- Use DL:
- <https://arxiv.org/pdf/1512.08183.pdf>

Task2- spam filtering

- If have a small dataset: use n-grams as feature indexes
- If have a huge dataset, it may not fit in memory. use hashing:
 $\text{hash}(\text{n-gram}) \% 2^{20}$
 - Has collections. But works in practice
 - `Sklearn.feature_extraction.text.HashingVectorizer`
 - Implemented in `vowpal wabbit` lib:
 - Popular ML lib for training linear models
 - Internally uses feature hashing, has lots of features, fast, scales well
 - You can give it a row text. It tokenizes it by white spaces. And use hash..
 - You can give it a text and hash number. (to be stored in that index of HT)

Hashing example

- **Example:** $\text{Hash}(s) = (\sum_{i=0}^{n=\text{len}(s)} s[i]p^i) \% 2^b$, p is a fixed prime number, b : bits of hash table
- $H(\text{good})=0$, $h(\text{not})=1$, $h(\text{movie})=2$, $h(a)=h(\text{did})=3$, $h(\text{like})=4$

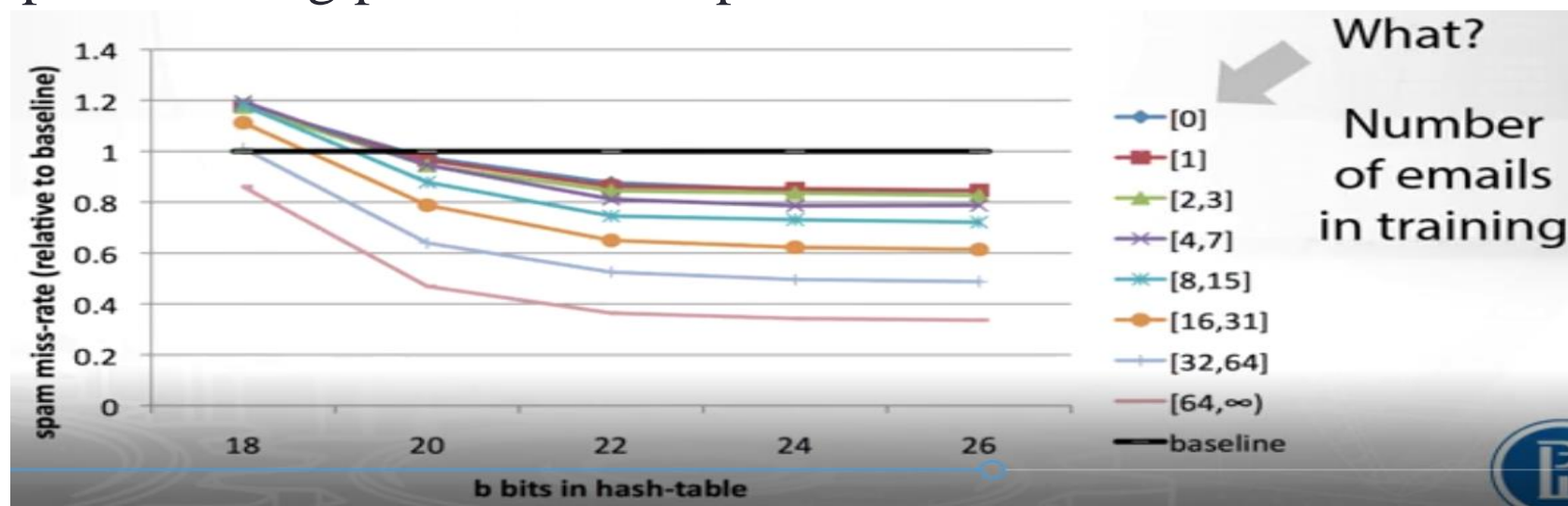


| | 0 | 1 | 2 | 3 | 4 |
|------------------|---|---|---|---|---|
| good movie | 1 | 1 | 0 | 0 | 0 |
| not a good movie | 1 | 1 | 1 | 1 | 0 |
| did not like | 0 | 0 | 1 | 1 | 1 |

- Why do we need hashing (huge data sets > use hashing to reduce training examples.. This seems same as using small ds?)
 - It can learn better models

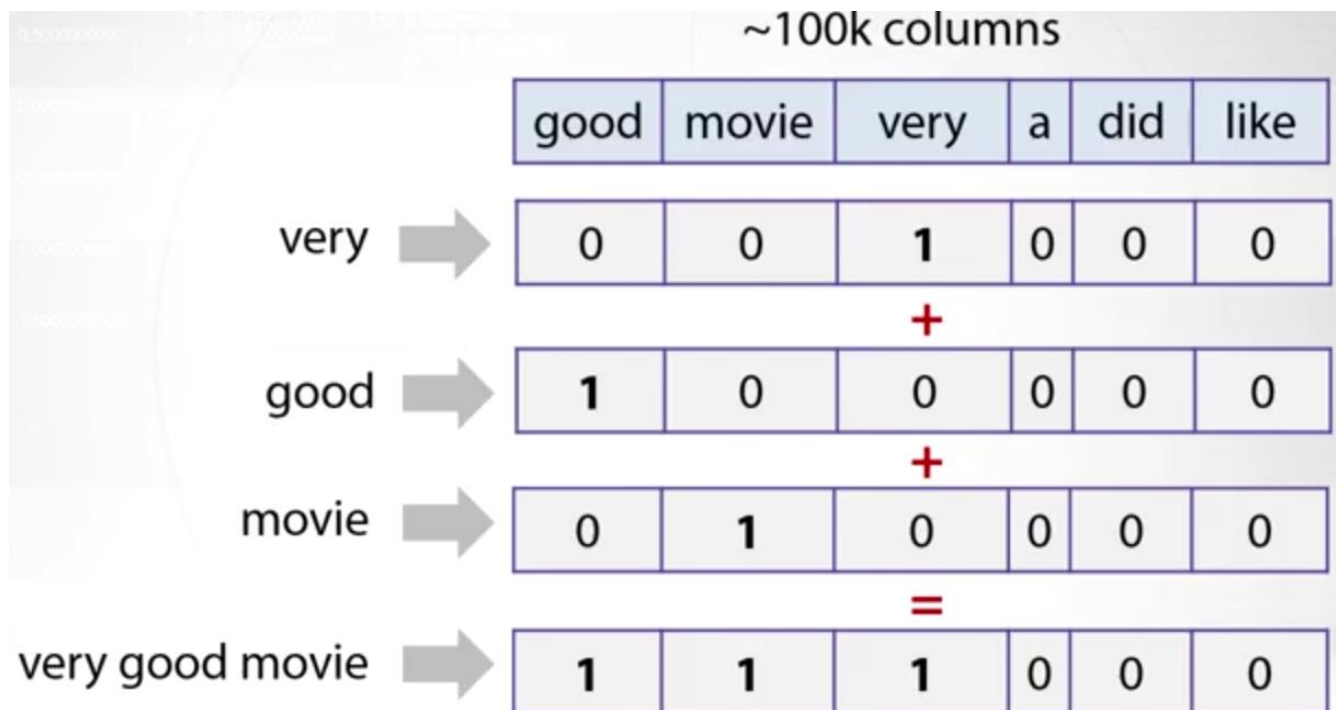
Personalized hashing

- Spam filtering.. User dependent...Personalized hashing: $h(\text{"user_token"})$ instead of $h(\text{"token"})$
- Why it works:
- 1- captures local user specific preferences
- 2-learn better global preferences (if a user has no emails in training set)
- <https://arxiv.org/pdf/1110.4198.pdf>



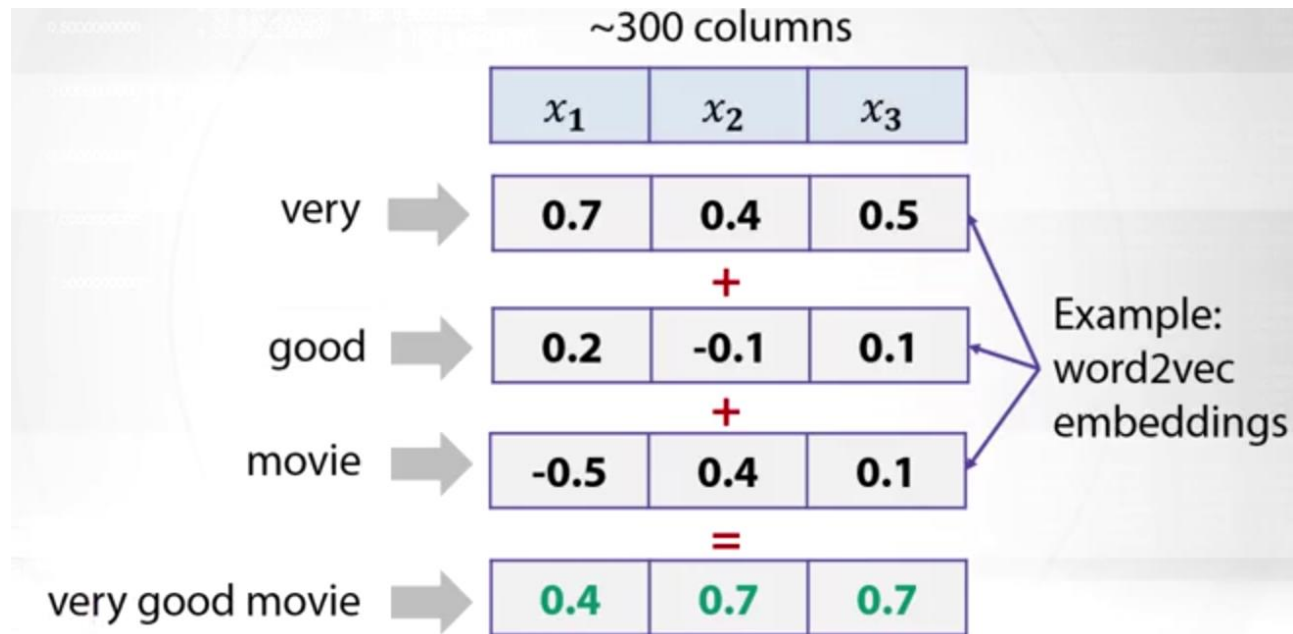
Bow representation

- Sum of sparse one-hot-encoded vectors



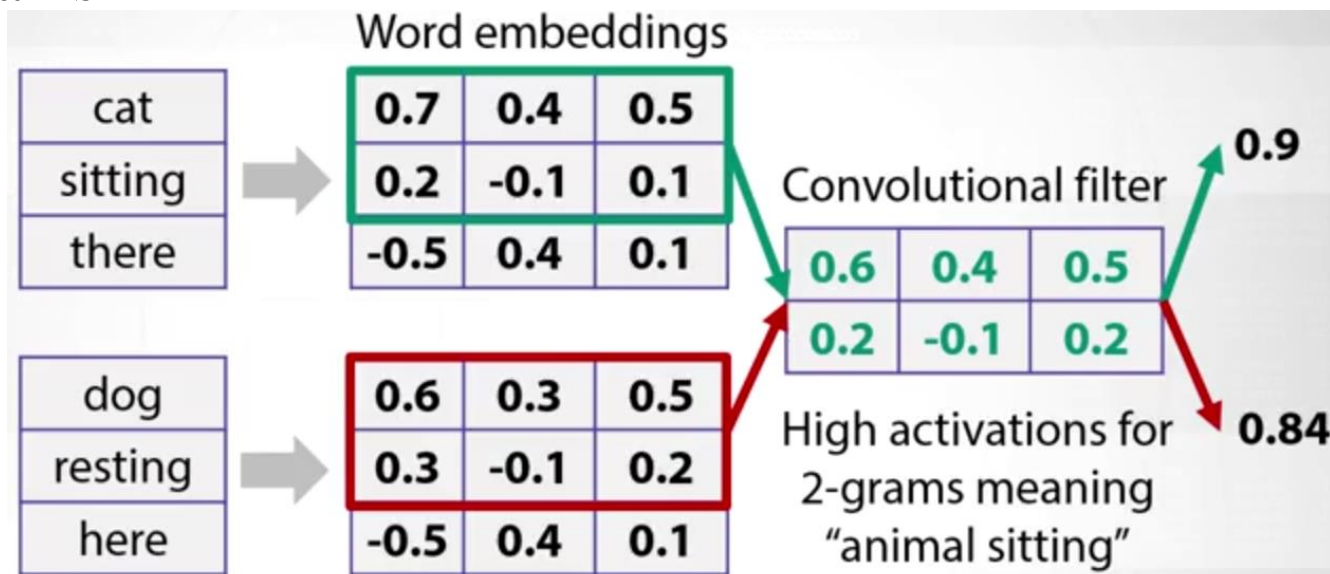
Dense neural way (word2vec rep)

- Words with similar context tend to have collinear vectors



1D convolution- better way

- 1D: because we slide window in only one direction (time)
- Like using n-grams instead of 1-grams
- And look at high level meaning of sentence
- Extended easily to higher n-grams, e.g. 3-gram, 4-grams
- Like CNN, one filter is not enough and must track many n-grams



| | | | |
|---------|------|------|------|
| | | | |
| Cat | 0.7 | 0.4 | 0.5 |
| Sitting | 0.2 | -0.1 | 0.1 |
| There | -0.5 | 0.4 | 0.1 |
| Or | -0.1 | 0.8 | -0.3 |
| here | -0.5 | 0.3 | 0.2 |
| | | | |

| |
|------------|
| 0.1 |
| 0.3 |
| -0.2 |
| 0.7 |
| -0.4 |

- Use padding (to have the same size as input sentence)
- Variable length of features (according to the sentence size)
- Loos ordering (where “cat sitting: appears)
- Use Max pooling over time-> get 0.7... do so for each filter

- 3,4,5-gram windows with 100 filters each.
- MLP on top of these 300 features
- <https://arxiv.org/pdf/1408.5882.pdf>
- <http://jmri.org/papers/volume12/collobert11a/collobert11a.pdf>

Character embedding

- Text as seq of chars..> easy to tokenize
- Use character n-grams

| | _ | c | a | t | _ | r | u | n | s | _ |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| One-hot encoded characters , length ~70 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 1 | 0 | ... | 0 | ... | ... | ... | ... | 0 |
| | ... | ... | ... | 1 | ... | 1 | 1 | 1 | 1 | ... |
| | 0 | 0 | 0 | ... | 0 | ... | ... | ... | ... | 0 |

- Use 1-D convolution with 1000 kernels, kernels widths 7,7,3,3,3,3
- Use pooling (max pool) with stride of 2 (to have position invariance) in each of 6 layers
- Apply MLP on 1000*34 matrix of features

Pooling
output

Another filter

Another filter

- Small DS: AG's news, sogou news, Dbpedia, yelp review polarity
- Large DS: Telp review full, yahoo answers, amazon review full, amazon review polarity

Errors on test set for classical models:

| Model | AG | Sogou | DBP. | Yelp P. | Yelp F. | Yah. A. | Amz. F. | Amz. P. |
|--------------|-------------|-------------|-------------|-------------|---------|---------|---------|---------|
| BoW | 11.19 | 7.15 | 3.39 | 7.76 | 42.01 | 31.11 | 45.36 | 9.60 |
| BoW TFIDF | 10.36 | 6.55 | 2.63 | 6.34 | 40.14 | 28.96 | 44.74 | 9.00 |
| ngrams | 7.96 | 2.92 | 1.37 | 4.36 | 43.74 | 31.53 | 45.73 | 7.98 |
| ngrams TFIDF | 7.64 | 2.81 | 1.31 | 4.56 | 45.20 | 31.49 | 47.56 | 8.46 |

Errors on test set for deep models:

| | | | | | | | | |
|--------------------|-------|------|------|------|--------------|--------------|--------------|-------------|
| LSTM | 13.94 | 4.82 | 1.45 | 5.26 | 41.83 | 29.16 | 40.57 | 6.10 |
| Sm. Full Conv. | 11.59 | 8.95 | 1.89 | 5.67 | 38.82 | 30.01 | 40.88 | 5.78 |
| Lg. Full Conv. Th. | 9.51 | - | 1.55 | 4.88 | 38.04 | 29.58 | 40.54 | 5.51 |
| Sm. Full Conv. Th. | 10.89 | - | 1.69 | 5.42 | 37.95 | 29.90 | 40.53 | 5.66 |
| Lg. Conv. | 12.82 | 4.88 | 1.73 | 5.89 | 39.62 | 29.55 | 41.31 | 5.51 |
| Sm. Conv. | 15.65 | 8.65 | 1.98 | 6.53 | 40.84 | 29.84 | 40.53 | 5.50 |
| Lg. Conv. Th. | 13.39 | - | 1.60 | 5.82 | 39.30 | 28.80 | 40.45 | 4.93 |
| Sm. Conv. Th. | 14.80 | - | 1.85 | 6.49 | 40.16 | 29.84 | 40.43 | 5.67 |