slidenumbers: true
footer: A. Ferrara. **Language models**. *Part 3: Statistical language models*. email, corse website, slack, github

Information Retrieval

[fit]Language models

Part 3: Statistical Language Models. Prof. Alfio Ferrara

Master Degree in Computer Science

Master Degree in Data Science and Economics

Skip-Gram Models (I)

The data sparsity problem is mitigated but not solved in **n-gram models** by smoothing or back-off techniques. Indeed, small variations in similar texts can have a large effect on the probability estimation in a n-gram model.

Example (from Casablanca)

- The train for Marseilles leaves at five
- A train to Marseilles and Lyon leaves five o'clock

The two sentences provide basically the same information (there's a train going to Marseilles at five), but they do not have any 2-gram nor 3-gram in common.

This means that a n-gram model will assign very different probabilities to those sentences.

Skip-Gram Models (II)

The problem with the previous sentences is partially due to the presence of *noisy* words that are not relevant for the sentence meaning but change the chain of n-grams (such as *the*, *a*, *at*, *Lyon*)

To solve this, **Skip-Gram models** introduce a notion of *word context* that is not limited to $n-1$ words as in n-grams, but that allow one to *skip k* words when indexing n-grams. The set S_{kn} of all the *k* skip *n* grams of a word sequence $w_1, \dots, w_n$ is then

$$S_k n = \left\{ w_{i1}, w_{i2}, \dots, w_{in} : \sum_{j=2}^{n} (i_j - i_{j-1} - 1 \leq k, i_j > i_{j-1} \forall j \right\}$$

Example

- The train for Marseilles leaves at five

```
def skip(sequence, n=2, s=2):
    k_grams = []
    for i in range(len(sequence)):
        for z in range(s):
            seq = [sequence[i]] + sequence[i+z+1:i+z+n]
            if len(seq) == n and seq not in k_grams:
                k_grams.append(tuple(seq))
    return k_grams

print(t1)
> ['the', 'train', 'for', 'marseilles', 'leaves', 'at', 'five']
skip(t1, n=2, s=3)
> [('the', 'train'), ('the', 'for'), ('the', 'marseilles'), ('train',
'for'),
   ('train', 'marseilles'), ('train', 'leaves'), ('for', 'marseilles'),
   ('for', 'leaves'), ('for', 'at'), ('marseilles', 'leaves'),
   ('marseilles', 'at'), ('marseilles', 'five'), ('leaves', 'at'),
   ('leaves', 'five'), ('at', 'five')]
```

Example

- The train for Marseilles leaves at five
- A train to Marseilles and Lyon leaves five o'clock

```
for e in [x for x in skip(t1, n=2, s=3)
            if x in skip(t2, n=2, s=3)]:
    print(e)

> ('train', 'marseilles')
> ('marseilles', 'leaves')
> ('leaves', 'five')
```

Continuous Bag of Words (CBOW) model

In the **Skip-gram model**, we aim at predicting the context of a word w given the word w, by collecting all the words *reachable* for w within a tolerance of k skip grams.

$$train \rightarrow (train, for), (train, marseilles), (train, leaves)$$

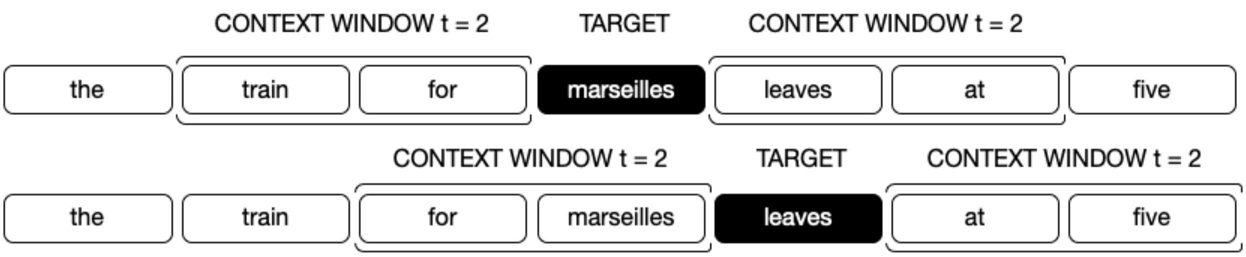
On the contrary, in the **Continuous Bag of Words model**, we aim at predicting a word w given the words appearing in its context

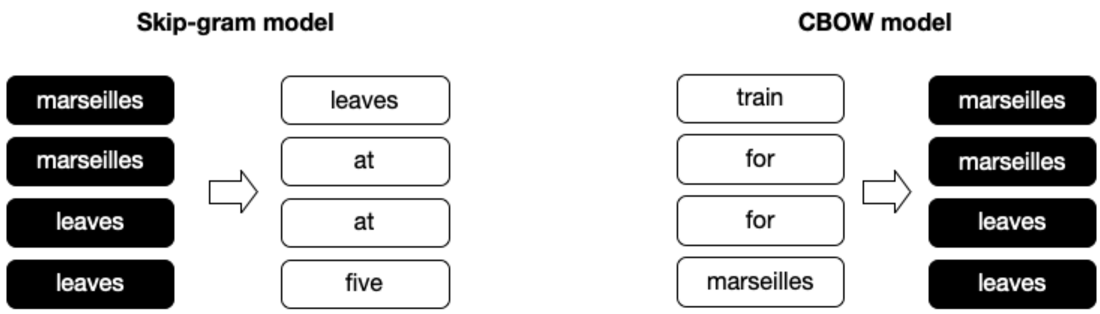
$(train, for), (train, marseilles), (train, leaves) \rightarrow train$

We will see more details about how to use **skip-gram** and **CBOW** models for learning in Part 4 of these lectures.

Word-Context models (I)

We can generalize the idea of exploiting the context of a word by computing a **word-context** matrix based on the counts of words appearing within t other words (before or after) the target word w.





Note that taking into account or not the word order (getting everything as a context or just words after) depends on the goals of the model:

- if we want to mimic the language (such as for generate text), order is relevant
- to focus on words embedding (and their semantics), order is less relevant

Word-Context models (II)

Having defined a notion of context, we can buil a context matrix $C = [c_{ij}]$ by counting for each word w_i how many times each word w_j occurs in its context.

A row (or column) of C can be used then as a **vector representation** of w_i (or w_j column-wise) in the space of the other words.

Example (Casablanca)

```python
from sklearn.metrics.pairwise import cosine_similarity

sigma = cosine_similarity(C, C)
j = list(C.index).index('train')
for i, x in sorted(enumerate(sigma[j]), key=lambda y: -y[1])[:5]:
    print(C.index[i])

> train, soon, immediately, living, sick
```

Matrix factorization

The obtained matrix $C = [c_{ij}]$ can be factorized into rank-p matrices (using for example **singular value decomposition (SVD)** to obtain a more compact representation as [1]:

$$C \approx UV^T$$

where the rows of U are the embedding we are looking for.

However, the factorization is still dominated by the zero terms. One option is to use stochastic gradient descent in order to change the objective function of factorization to de-emphasize the zero entries by sampling zero entries at a lower rate. Such an approach results in **weighted matrix factorization**.

Weighted matrix factorization

Element-wise, the decomposition problem can be written as:

$$c_{ij} \approx \hat{c}_{ij} = u_i \cdot v_j,$$

where u_i and v_j are the ith and jth rows of matrices U and V, respectively. The goal of gradient descent is to estimate u_i, v_j, and bias parameters b_i and b_j by **minimizing**

$$\sum_i \sum_j (c_{ij} - b_i - b_j - u_i \cdot v_j)^2$$

where the error rate is $e_{ij} = c_{ij} - \hat{c}_{ij}$

Stochastic gradient descent

Initially, U and V are randomly initialized. In subsequent iterations U and V are updated for some randomly selected entries (i, j) in C according to

$$u_i \leftarrow u_i(1-\alpha) + \alpha e_{ij} v_j$$

$$v_i \leftarrow v_i(1-\alpha) + \alpha e_{ij} u_j$$

$$b_i \leftarrow b_i(1-\alpha) + \alpha e_{ij}$$

$$b_j \leftarrow b_j(1-\alpha) + \alpha e_{ij}$$

where α is the learning rate. Since the update is not excuted for all the matrix entires, we can sample the entries in order to reduce the impact of zeros.

Negative sampling

A cycle of stochastic gradient descent samples through all the non-zero entries of C but only a random sample of zero entries.

The random sample size is always k times the number of non-zero entries.

The value of $k > 1$ is a user-driven parameter, and it implicitly controls the weight of positive and negative samples in the factorization.

GloVe embedding

In order to reduce the impact of the high variability if the words frequency, the **GloVe (Global Vectors for Word Representation)** method [2] introduces two variations in the factorization process:

1. A dumping factor $\log(1 + c_{ij})$ is used instead of c_{ij} to reduce the impact of highly frequent words
2. The error associated with an entry (i, j) is weighted using a threshold M and a parameter α

$weight(i, j) = \min\left\{1, \frac{c_{ij}}{M}\right\}^{\alpha}$

Topic Modeling

Latent Dirichlet allocation

Latent Dirichlet allocation (LDA) is a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of **hidden variables**, called **topics**. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities.

Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, *55*(4), 77-84.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, *3*(Jan), 993-1022.

The generative model of LDA

Let be $\phi^{(k)}$ a discrete probability distribution over the vocabulary for the k_{th} topic

Let θ_d be a document distribution over the topics

Let z_i be the topic index for the word w_i

- For $k = 1, \ldots, K \rightarrow \phi^{(k)} \sim Dir(\beta)$
    - For each $d \in D \rightarrow \theta_d \sim Dir(\alpha)$
        - Generate each word $w_i \in d$: $z_i \sim Discrete(\theta_d)$ $w_i \sim Discrete(\phi^{(z_i)})$

Derivation of the joint distribution

$p(w, z, \theta, \phi \mid \alpha, \beta) = p(\phi \mid \beta)p(\theta \mid \alpha)p(z \mid \theta)p(w \mid \phi_z)$

Where, starting from data, we need to estimate z, ϕ, and θ.

- θ_d is a representation of d in the topic space
- z_i represents which topic generated the word w_i
- Each $\phi^{(k)}$ is a matrix $K \times W$ where $\phi_{ij} = p(w_i \mid z_j)$

In order to find the latent variables, we need to solve

$p(\theta, \phi, z \mid w, \alpha, \beta) = \frac{p(\theta,\phi,z,w \mid \alpha,\beta)}{p(w \mid \alpha,\beta)}$

Gibbs sampling

Gibbs Sampling is one member of a family of algorithms from the Markov Chain Monte Carlo (MCMC) framework

MCMC algorithms aim to construct a Markov chain that has the target posterior distribution as its stationary distribution. In other words, after a number of iterations of stepping through the chain, sampling from the distribution should converge to be close to sampling from the desired posterior.

Example

To sample from $p(X) = p(x_1, \ldots, x_n)$ we can:

Randomly initialize X. Then, for each iterative step t

$x_1^{t+1} \sim p(x_1 \mid x_2^t, x_3^t, \ldots x_n^t)$

$x_2^{t+1} \sim p(x_2 \mid x_1^{t+1}, x_3^t, \ldots x_n^t)$

$\ldots$

$x_n^{t+1} \sim p(x_n \mid x_1^{t+1}, x_2^{t+1}, \ldots x_{n-1}^{t+1})$

Gibbs sampling for LDA

We need to estimate $\phi^{(k)}$, θ_d and z_i. But note that

$\theta_{d,z_i} = \frac{n(d,z_i)+\alpha}{\sum\limits_{j \in Z} n(d,z_j)+\alpha},$

where $n(d, z)$ is the number of times document d is assigned to topic z

$$\phi_{z,w_i} = \frac{n(z,w_i)+\beta}{\sum\limits_{j \in W} n(z,w_j)+\beta},$$

where $n(z, w)$ is the number of times word w is assigned to topic z

Collapsed Gibbs sampling

The previous equations require to just estimate z_i

$$p(z_i \mid z_{\neg i}, \alpha, \beta, w),$$

where $z_{\neg i}$ represents all topic allocations **except** for z_i. This results in

$$p(z_i \mid z_{\neg i}, \alpha, \beta, w) = \frac{p(z_i, z_{\neg i}, w \mid \alpha, \beta)}{p(z_{\neg i}, w \mid \alpha, \beta)} \propto p(z_i, z_{\neg i}, w \mid \alpha, \beta) = p(z, w \mid \alpha, \beta)$$

For a detailed discussion on how $p(z, w \mid \alpha, \beta)$ can be estimated, see this introduction to topic modeling

Example (I)

To have a meaningfull example, we need some more documents

```
r = {'$match': {'character.movie.genres': 'war'}}
tm = MovieDialogCollection(db_name, collection,
                           use_pos=False,
                           drop_stopwords=True,
                           pipeline=[r, p, s],
                           pos_filter=['NOUN'],
                           lemma=True)
```

Example (II)

```
from sklearn.decomposition import LatentDirichletAllocation

lda = LatentDirichletAllocation(n_components=50)
I = defaultdict(lambda: defaultdict(lambda: 0))
docs = []
for doc, tokens in tm.get_tokens():
    docs.append(doc)
    for token in tokens:
        I[doc][token] += 1
X = pd.DataFrame(I).T
X.fillna(0, inplace=True)
X.shape
> (7073, 4244)
```

Example (III)

Document distribution over documents

```
theta = lda.fit_transform(X)
theta.shape
> (7073, 50)
```

Word distribution over topics

```
phi = lda.components_ / lda.components_.sum(axis=1)[:, np.newaxis]
phi.shape
> (50, 4244)
```

Example (IV)

Get most relevant documents per topic

```
documents = dict([(d, t) for d, t in tm])
topic = 1
for i, x in sorted(enumerate(theta[:,topic]), key=lambda y: -y[1])[:5]:
    print(i, x)
    print(documents[docs[i]], '\n')
>
4192 0.859999999999999
What do you do in the joint besides pimp?

6874 0.836666666666466
She'll never forgive me!

```

```
4222 0.8203732644422244
Let's get his clothes off quick.

4999 0.8039999999999722
I sent telegrams, I guess the military traffic held them up.

7042 0.8039999999999508
You will remember the name? Von Scherbach? VON SCHER-BACH!

```

Example (V)

Get most relevant words per topic

```
for i, x in sorted(enumerate(phi[topic,:]), key=lambda y: -y[1])[:5]:
    print(i, x)
    print(X.columns[i], '\n')
>
37 0.347526097853741
sir

40 0.05967234120950544
lord

271 0.04346697422513307
thank

431 0.02282598214856176
lady

771 0.022410885193540186
tree
```

1. For matrix factorization and SVD have a look at Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge university press. Chapter 18 ↩

2. Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543). ↩