

Feature extraction from texts and images

Solely text/images competitions



For example, we can use search engines in order to find similar text.

Feature extraction from texts and images

Common features + text

Titanic dataset

A1 ▼ f_{sc} survived						
	A	B	C	D	E	F
1	survived	pclass	name	sex	age	sibsp
2	0	3	Braund, Mr. Owen Harris	male	22	1
3	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1
4	1	3	Heikkinen, Miss. Laina	female	26	0
5	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1
6	0	3	Allen, Mr. William Henry	male	35	0
7	0	3	Moran, Mr. James	male		0
8	0	1	McCarthy, Mr. Timothy J	male	54	0
9	0	3	Palsson, Master. Gosta Leonard	male	2	3
10	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0
11	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1
12	1	3	Sandstrom, Miss. Marguerite Rut	female	4	1
13	1	1	Bonnell, Miss. Elizabeth	female	58	0

Feature extraction from texts and images

Common features + images/text



TRADESHIFT®

When feature extraction is done, we can treat extracted features differently. Sometimes we just want to add new features to existing dataframe. Sometimes we even might want to use the right features independently, and in end, make stake in with the base solution.

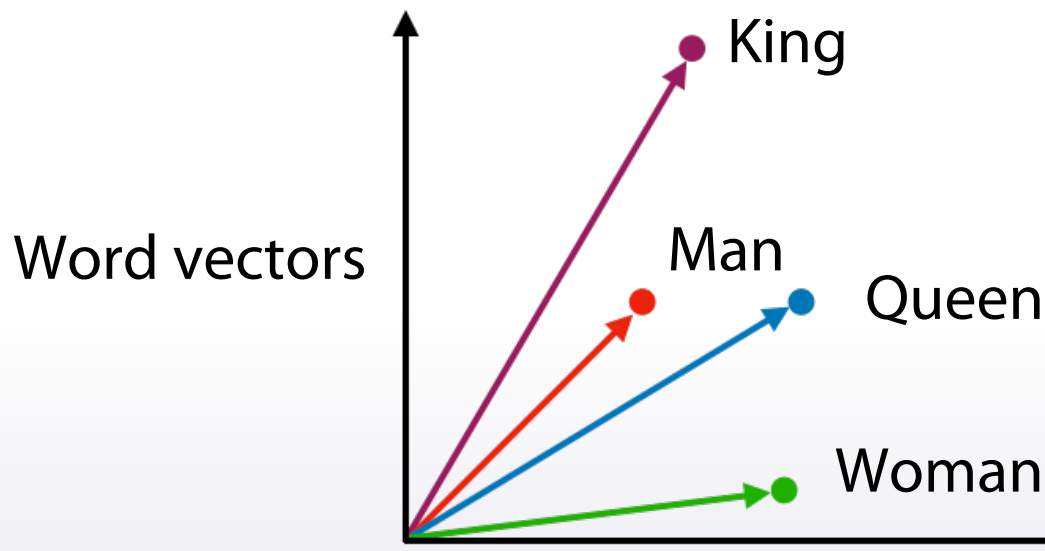
Text -> vector

Let's start with featured extraction from text. There are two main ways to do this. First is to apply bag of words, and second, use embeddings like word to vector.

1. Bag of words:



2. Embeddings (~word2vec):



Bag of words

(excited) Hi everyone!	I'm so excited about this course!	So excited. SO EXCITED. EXCITED, I AM!
---------------------------	---	--

CountVectorizer



hi	every one	I'm	so	excited	about	this	course
1	1			1			
		1	1	1	1	1	1
		1	2	3			

```
sklearn.feature_extraction.text.CountVectorizer
```

Bag of words: TFiDF

Bag of words: TFiDF

Term frequency

```
tf = 1 / x.sum(axis=1)[:,None]
```

```
x = x * tf
```

One way to achieve the first goal of making a sample small comparable is to normalize sum of values in a row. In this way, we will count not occurrences but frequencies of words. Thus, texts of different sizes will be more comparable. This is the exact purpose of term frequency transformation

Bag of words: TFiDF

Term frequency

```
tf = 1 / x.sum(axis=1)[:,None]
```

```
x = x * tf
```

Inverse Document Frequency

```
idf = np.log(x.shape[0] / (x > 0).sum(0))
```

```
x = x * idf
```

To achieve the second goal, that is to boost more important features, we'll make post process our matrix by normalizing data column wise. A good idea is to normalize each feature by the inverse fraction of documents, which contain the exact word corresponding to this feature.

Bag of words: TFiDF

Term frequency

```
tf = 1 / x.sum(axis=1)[:,None]
```

```
x = x * tf
```

Inverse Document Frequency

```
idf = np.log(x.shape[0] / (x > 0).sum(0))
```

```
x = x * idf
```

```
sklearn.feature_extraction.text.TfidfVectorizer
```

In this case, features corresponding to frequent words will be scaled down compared to features corresponding to rarer words.

4:01

We can further improve this idea by taking a logarithm of these numberization coefficients. As a result, this will decrease the significance of widespread words in the dataset and do require feature scaling. This is the purpose of inverse document frequency transformation.

Bag of words: TF

(excited) Hi everyone!	I'm so excited about this course!	So excited. SO EXCITED. EXCITED, I AM!
---------------------------	---	--



hi	every one	I'm	so	excited	about	this	course
0.33	0.33			0.33			
		0.16	0.16	0.16	0.16	0.16	0.16
		0.16	0.33	0.5			

Bag of words: TF+idf

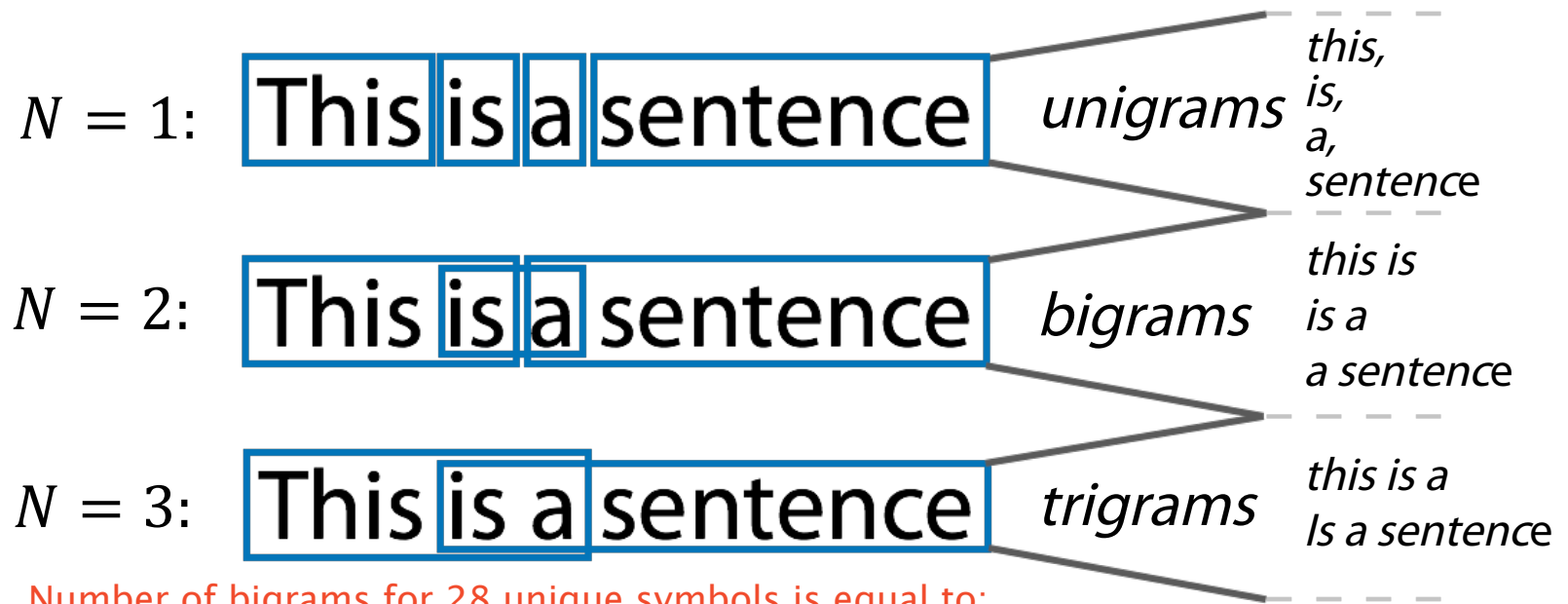
(excited) Hi everyone!	I'm so excited about this course!	So excited. SO EXCITED. EXCITED, I AM!
---------------------------	---	--



hi	every one	I'm	so	excited	about	this	course
0.36	0.36			0			
		0.06	0.06	0	0.18	0.18	0.18
		0.06	0.13	0			

N-grams

The concept of Ngram is simple, you add not only column corresponding to the word, but also columns corresponding to inconsequent words.



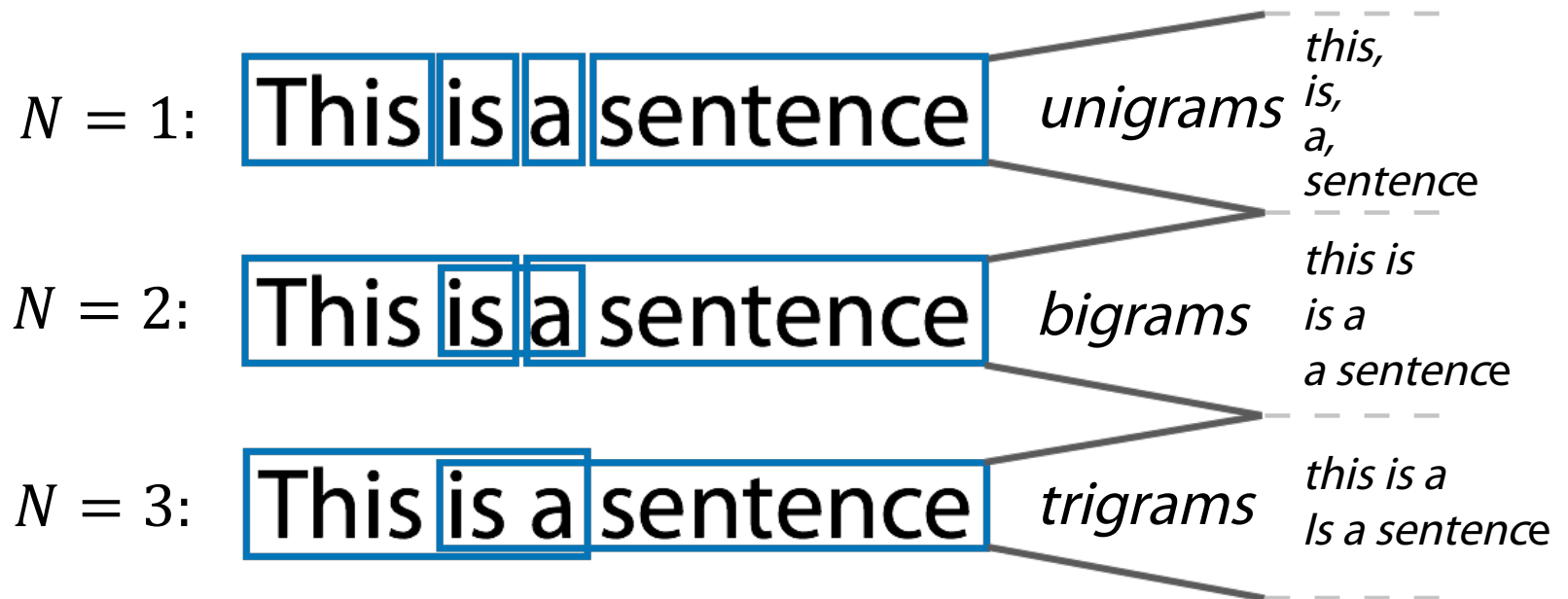
Number of bigrams for 28 unique symbols is equal to:

$$28 * 28 = 784$$

Correct

If you have 28 unique symbols, the number of all possible combinations is equal to $28 * 28$.

N-grams



```
sklearn.feature_extraction.text.CountVectorizer:  
Ngram_range, analyzer
```

Texts preprocessing

1. Lowercase
2. Lemmatization
3. Stemming
4. Stopwords

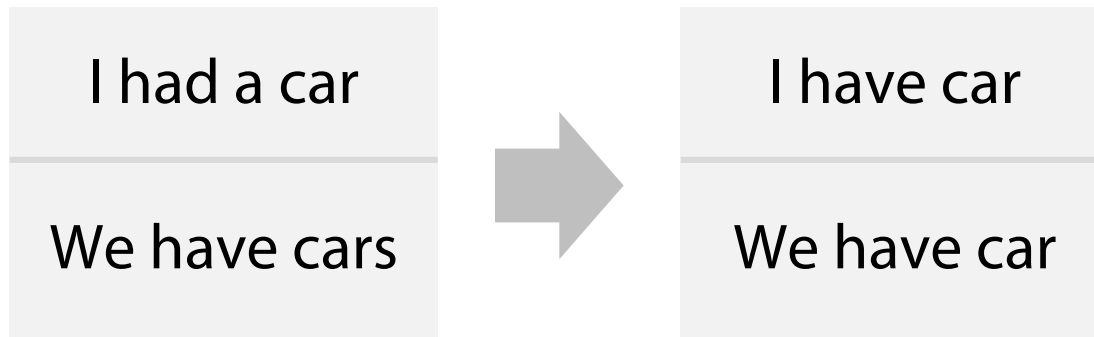
Texts preprocessing: lowercase

Very, very sunny.
Sunny... Sunny!



Very	very	Sunny	sunny
1	1	0	1
0	0	2	0

Texts preprocessing: lemmatization and stemming



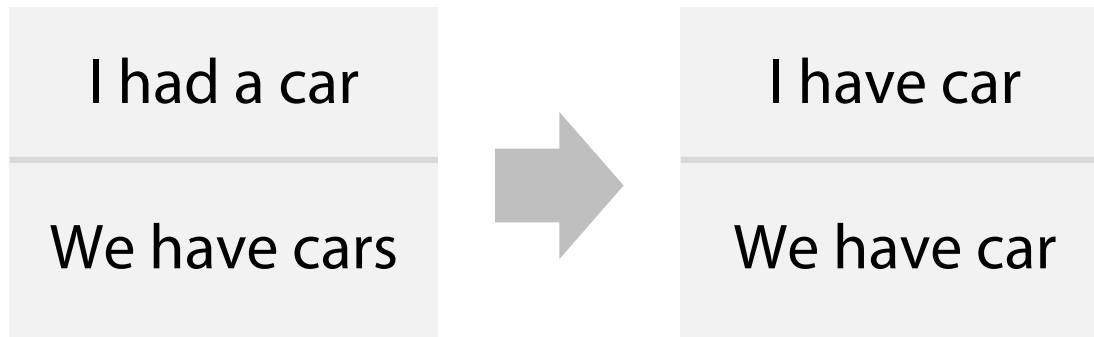
Stemming:

democracy, democratic, and democratization -> democr

Lemmatization:

democracy, democratic, and democratization -> democracy

Texts preprocessing: lemmatization and stemming



Stemming:

democracy, democratic, and democratization -> democr

Saw -> s

Lemmatization:

democracy, democratic, and democratization -> democracy

Saw -> see or saw (depending on context)

Texts preprocessing: stopwords

Examples:

1. Articles or prepositions
2. Very common words

NLTK, Natural Language Toolkit library for python

```
sklearn.feature_extraction.text.CountVectorizer:  
max_df
```

Conclusion

Pipeline of applying BOW

1. Preprocessing:

Lowercase, stemming, lemmatization, stopwords

2. Bag of words:

Ngrams can help to use local context

3. Postprocessing: TFiDF