

# Классификация текстов и анализ тональности

**Классификация текстов** — соотнесение документа к одной из нескольких категорий. Применяется в:

- разделения веб страниц и сайтов по тематическим каталогам;
- борьбы со спамом;
- определение языка текста и т.д.

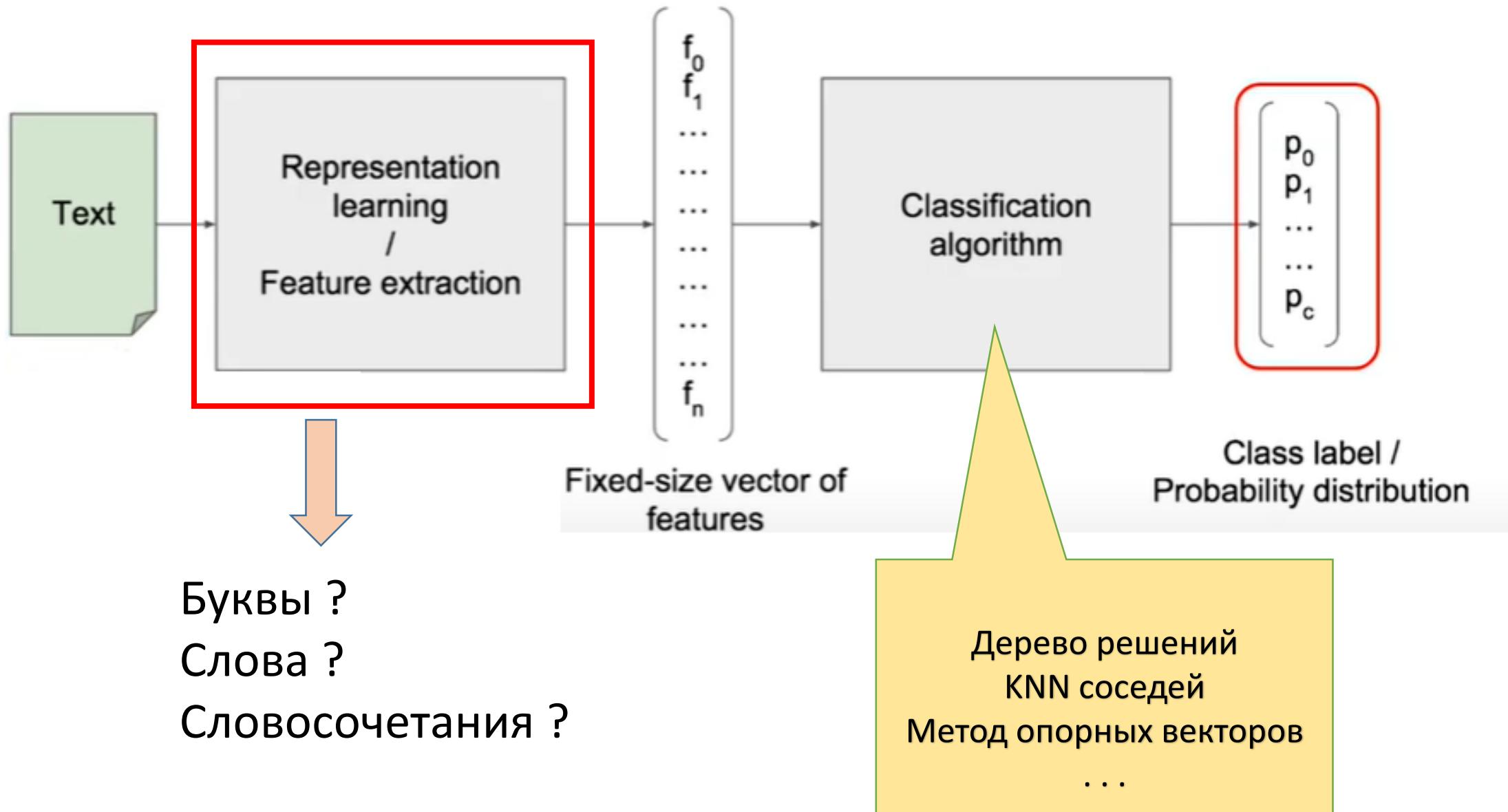
**Анализ тональности текста** (англ. Sentiment analysis) — определение эмоциональной окраски (тональности) текста. В общем случае, задача анализа тональности текста эквивалентна задаче классификации текста, где категориями текстов могут быть тональные оценки. Примеры тональных оценок:

- позитивная;
- негативная;
- нейтральная.

# Классификация намерений

<b>узнать_имя</b>	<p>как тебя зовут? скажи свое имя с кем я общаюсь? я Ксюша, а ты?</p> <p>...</p>
<b>откуда_номер</b>	<p>откуда вы мой номер узнали вы где мой телефон взяли? кто вам дал этот номер? откуда у вас этот номер телефона</p> <p>...</p>

# Text classification in general



## Векторное представление текста

### Bag of words (BOW)

1. the red dog →
2. cat eats dog →
3. dog eats food →
4. red cat eats →

the	red	dog	cat	eats	food
1	1	1	0	0	0
0	0	1	1	1	0
0	0	1	0	1	1
0	1	0	1	1	0

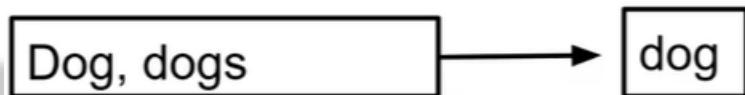
Problems:

- No information about words order
- Word vectors are huge and very sparse
- Word vectors are not normalized
- Same words can take different forms

Используя алгоритмы вроде Bag of Words, мы теряем порядок слов в тексте, а значит, тексты "i have no cows" и "no, i have cows" будут идентичными после векторизации, хотя и противоположными семантически.

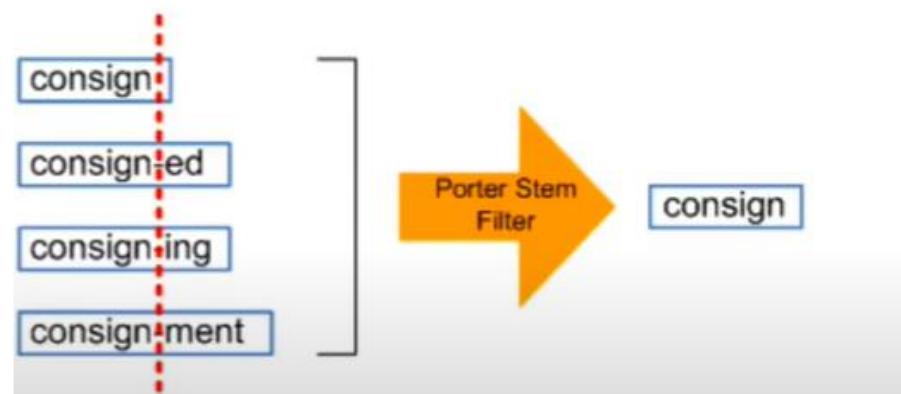
# Предобработка текста (Preprocessing)

Token normalization



Нормализовать токены (слова) можно с помощью стемминга и лемматизации

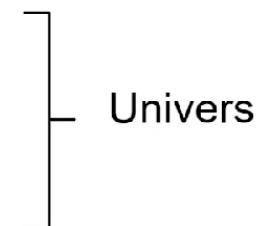
- **Stemming:** removing and replacing suffixes to get to the root of the word (**stem**)



Проблемма стемминга

Overstemming

- University
- Universal
- Universities
- Universe



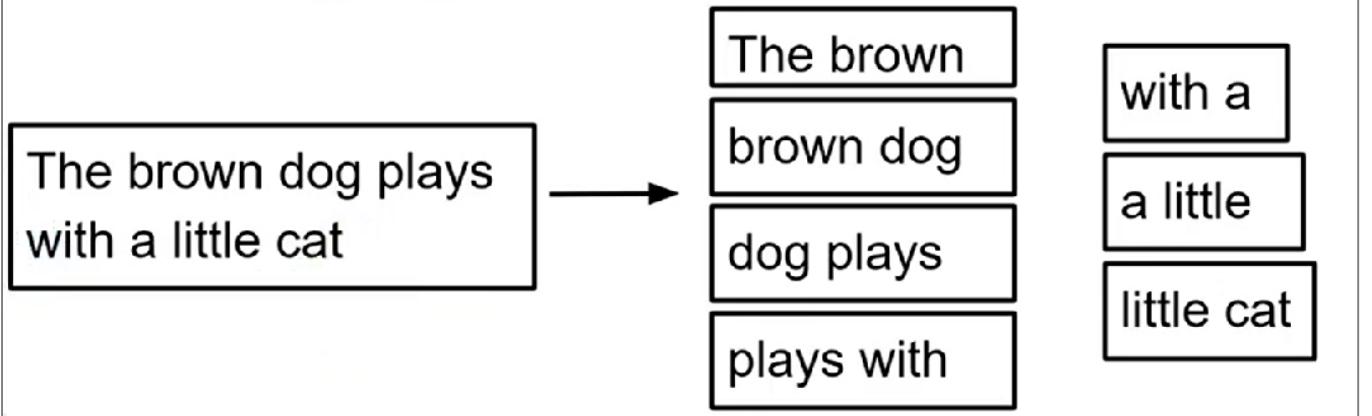
СПИСОК ЛЕММАТИЗАЦИИ

глаз	глаза	тут	тута
глаз	глазу	тут	туту
глаз	глазом	тут	тутом
глаз	глазе	тут	туте
глаз	глазам	тут	туты
глаз	глазами	тут	тутов
глаз	глазах	тут	тутам

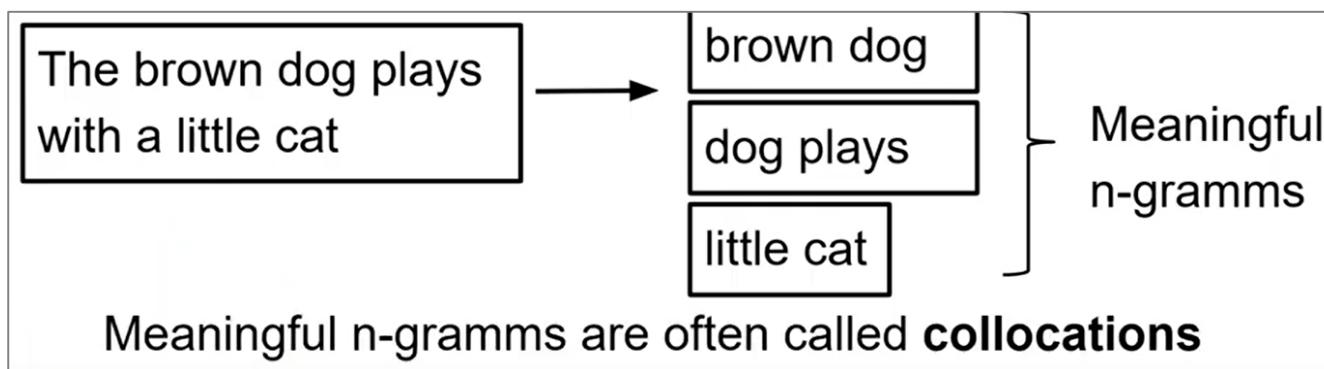
- **Lemmatization:** to get base or dictionary form of a word (**lemma**) Based on **WordNet** database

# Bag of words (BOW)

- How to improve BOW?
  - Use n-gramms instead of words!



Удаляем n-gramms по  
правилам



Потерян порядок слов  
Как исправить?

- Использовать n-граммы  
n-грамма — это последовательность из n токенов (слов), идущих в тексте подряд.

Delete:

- High-frequency n-gramms
  - Articles, prepositions
  - Auxiliary verbs (to be, to have, etc.)
  - General vocabulary
- Low-frequency n-gramms
  - Typos
  - Combinations that occur 1-2 times in a text

## Потерян порядок слов

### Как исправить?

- Использовать контекст

### Source Text

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

The quick brown fox jumps over the lazy dog. →

### Training Samples

(the, quick)  
(the, brown)

(quick, the)  
(quick, brown)  
(quick, fox)

(brown, the)  
(brown, quick)  
(brown, fox)  
(brown, jumps)

(fox, quick)  
(fox, brown)  
(fox, jumps)  
(fox, over)

TF-IDF (от англ. TF — term frequency, IDF — inverse document frequency) — статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

В метрику TF-IDF входят два сомножителя: TF и IDF.

TF (*term frequency* — частота слова) — отношение числа вхождений некоторого слова к общему числу слов документа. Таким образом, оценивается важность слова  $t_i$  в пределах отдельного документа.

$$\text{tf}(t, d) = \frac{n_t}{\sum_k n_k} ,$$

где  $n_t$  есть число вхождений слова  $t$  в документ, а в знаменателе — общее число слов в данном документе.

**IDF** (inverse document frequency — обратная частота документа) — инверсия частоты, с которой некоторое слово встречается в документах коллекции.

Учёт IDF уменьшает вес **широкоупотребительных слов** (частиц, предлогов и других служебных частей речи). Для каждого уникального слова в пределах конкретной коллекции документов существует только одно значение IDF.

$$\text{idf}(t, D) = \log \frac{|D|}{|\{d_i \in D \mid t \in d_i\}|}, [2]$$

где

- $|D|$  — число документов в коллекции;
- $|\{d_i \in D \mid t \in d_i\}|$  — число документов из коллекции  $D$ , в которых встречается  $t$  (когда  $n_t \neq 0$ ).

Таким образом, мера TF-IDF является произведением двух сомножителей:

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

Большой вес в TF-IDF получат слова с высокой **частотой** в пределах конкретного документа и с низкой частотой употреблений в других документах.

*Sentence A: The car is driven on the road.*

*Sentence B: The truck is driven on the highway.*

(each sentence is a separate document)

```
from sklearn.feature_extraction.text
import TfidfVectorizer
```

Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	$\log(2/2)=0$	0	0
Car	1/7	0	$\log(2/1)=0.3$	0.043	0
Truck	0	1/7	$\log(2/1)=0.3$	0	0.043
Is	1/7	1/7	$\log(2/2)=0$	0	0
Driven	1/7	1/7	$\log(2/2)=0$	0	0
On	1/7	1/7	$\log(2/2)=0$	0	0
The	1/7	1/7	$\log(2/2)=0$	0	0
Road	1/7	0	$\log(2/1)=0.3$	0.043	0
Highway	0	1/7	$\log(2/1)=0.3$	0	0.043

Вес слова

Мера TF-IDF часто используется для представления документов коллекции в виде числовых векторов, отражающих важность использования каждого слова из некоторого набора слов (количество слов набора определяет размерность вектора) в каждом документе. Подобная модель называется векторной моделью и даёт возможность сравнивать тексты, сравнивая представляющие их вектора в какой-либо метрике

## Визуализация метрики ТF

term weighting  none term frequency raw document frequency inverse document frequency TF-IDF  
stopwords grayed out

charles bailey WAS indicted for feloniously stealing On **the** 29th of december two dressed deer skins value 20 S **the** property of samuel savage **and** richard savage  
richard savage **i** am a leather seller 63 chiswell street my partner S name is samuel savage a few days previous to **the** 29th of december **i** looked out seventy  
skins for an order these skins being **of** a bad colour **i** directed them to be brimstoned to make them of equal colour pale On **the** 29th in the afternoon  
**i** saw them all smooth On a horse a few hours afterwards they appeared very much tumbled **and** one **was** thrown into **the** yard **and** dirtied **i** caused  
them to be brought in **the** warehouse **and** counted there **was** two gone our foreman went to worship street **and** brought armstrong **and**  
vickrey they searched **and** found this skin in **the** prisoner S breeches **and** the other skin **was** found in **the** workshop carter **i** am  
foreman to samuel **and** richard savage **the** seventy skins **i** was with mr savage looking them out **i** took them out of **the** stove **and**  
counted them on **the** horse **and** on friday **i** counted them three times over there were no more than sixty eight instead of seventy **i** went to worship  
street brought mr armstrong **and** vickery with me they waited till **the** men left work **and** when they came down they were searched **and** on  
**the** prisoner one skin **was** found john armstrong **i** went to this gentleman S house after **the** men came down vickrey **and** i were searching in  
one minute vickrey called me **i** received this skin from him it **WAS** taken out of **the** prisoner S breeches **i** have had it ever since john vickrey q you were

## Визуализация метрики IDF



# Визуализация метрики TF-IDF



**Метод вложения слов** - это метод обработки естественного языка, который используется для представления слов в среде глубокого обучения.

Основное преимущество использования вложения слов состоит в том, что **оно позволяет группировать слова схожего контекста, а разные слова располагать далеко друг от друга**. Это делается с помощью матрицы вложения. Сходство двух слов можно найти с помощью косинусного сходства.

Матрица вложения:

	King	Queen	Princess	Boy
Royal	0,99	0,99	0,99	0,01
Male	0,99	0,02	0,01	0,98
Female	0,02	0,99	0,99	0,01
Age	0,7	0,6	0,1	0,2
⋮	⋮	⋮	⋮	⋮

Два наиболее популярных алгоритма вложения слов WORD2VEC, и GLOVE позволяют нам представлять слово в виде вектора (часто называемого вложением). Эти алгоритмы выявляют семантическое сходство слов и отражают различные аспекты значения слова .

Они используются во многих приложениях НЛП, таких как анализ настроений, кластеризация документов, ответы на вопросы, обнаружение перефразирования и так далее. Предварительно обученные модели для обоих этих внедрений легко доступны, и их легко включить в код Python.

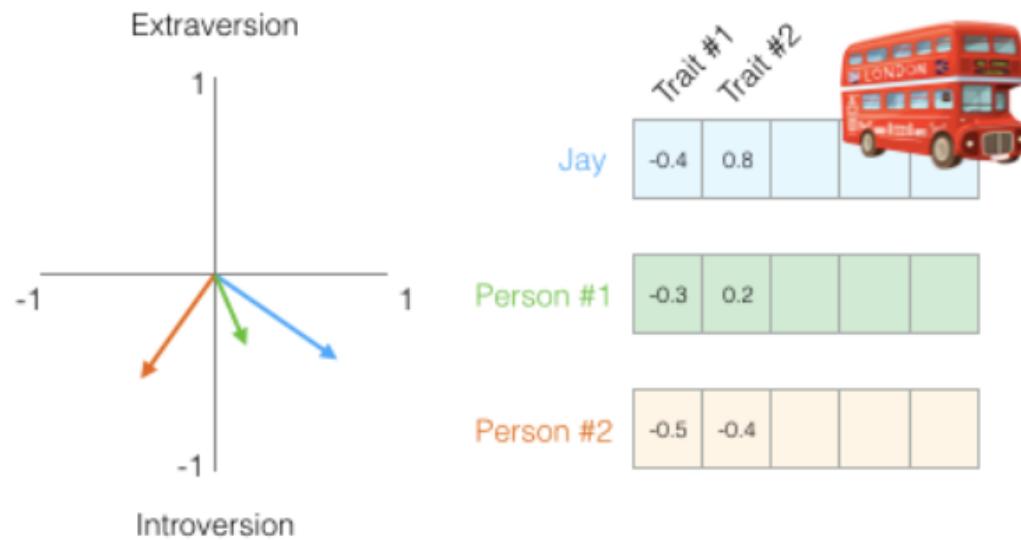
Концепция вложений (embeddings) — одна из самых замечательных идей в машинном обучении. Если вы когда-нибудь использовали Siri, Google Assistant, Alexa, Google Translate или даже клавиатуру смартфона с предсказанием следующего слова, то уже работали с моделью обработки естественного языка на основе вложений. За последние десятилетия произошло значительное развитие этой концепции для нейронных моделей (последние разработки включают контекстуализированные вложения слов в передовых моделях, таких как BERT и GPT).

Рассмотрим концепцию и механику генерации вложений с помощью Word2vec. Начнём с примера, чтобы ознакомиться с тем, как представлять объекты в векторном виде. Вы знаете, насколько много о вашей личности может сказать список из пяти чисел (вектор)?

По шкале от 0 до 100 у вас интровертный или экстравертный тип личности (где 0 — максимально интровертный тип, а 100 — максимально экстравертный)? Вы когда-нибудь проходили личностный тест: например, MBTI, а ещё лучше «большую пятёрку»? Вам дают список вопросов, а затем оценивают по нескольким осям, в том числе интровертность/экстравертность.

Openness to experience	.....	79 out of 100
Agreeableness	.....	75 out of 100
Conscientiousness	.....	42 out of 100
Negative emotionality	.....	50 out of 100
Extraversion	.....	58 out of 100

Теперь можно сказать, что этот вектор частично отражает мою личность. Это полезное описание, если сравнить разных людей. Допустим, меня сбил красный автобус, и нужно заменить меня похожей личностью. Кто из двух людей на следующем графике больше на меня похож?



При работе с векторами сходство  
обычно вычисляется по коэффициенту  
Отии (геометрический коэффициент)



$$\text{cosine\_similarity}(\begin{matrix} -0.4 & 0.8 \end{matrix}, \begin{matrix} -0.3 & 0.2 \end{matrix}) = 0.87$$



$$\text{cosine\_similarity}(\begin{matrix} -0.4 & 0.8 \end{matrix}, \begin{matrix} -0.5 & -0.4 \end{matrix}) = -0.20$$

Вывод Person 1 подходит больше

Теперь разберемся в механизме вложения слов, для этого познакомимся с понятием:

## Дистрибутивная (распределительная) семантика.

[https://lena-voita.github.io/nlp\\_course/word\\_embeddings.html#main\\_content](https://lena-voita.github.io/nlp_course/word_embeddings.html#main_content)

Чтобы зафиксировать значение слов в их векторах, нам сначала нужно определить понятие значения, которое можно использовать на практике. Попробуем понять, **как мы, люди, узнаем, какие слова имеют схожее значение.**

How to: go over the slides at your pace. Try to notice how your brain works.

Do you know what the word **tezgüino** means ?

(We hope you do not)



Now look how this word is used in different contexts:

A bottle of **tezgüino** is on the table.

Everyone likes **tezgüino**.

**Tezgüino** makes you drunk.

We make **tezgüino** out of corn.



Can you understand what **tezgüino** means ?



Now look how this word is used in different contexts:

A bottle of **tezgüino** is on the table.

Everyone likes **tezgüino**.

**Tezgüino** makes you drunk.

We make **tezgüino** out of corn.



**Tezgüino** is a kind of alcoholic beverage made from corn.



With context, you can understand the meaning!





How did you do this?



- (1) A bottle of \_\_\_\_\_ is on the table.
- (2) Everyone likes \_\_\_\_\_.
- (3) \_\_\_\_\_ makes you drunk.
- (4) We make \_\_\_\_\_ out of corn.

What other words fit  
into these contexts ?



(1) A bottle of \_\_\_\_\_ is on the table.

(2) Everyone likes \_\_\_\_\_.

(3) \_\_\_\_\_ makes you drunk.

(4) We make \_\_\_\_\_ out of corn.

What other words fit  
into these contexts ?



	(1)	(2)	(3)	(4)	...	← contexts
tezgüino	1	1	1	1		
loud	0	0	0	0		← rows show contextual
motor oil	1	0	0	1		properties: 1 if a word can
tortillas	0	1	0	1		appear in the context, 0 if not
wine	1	1	1	0		



Это **гипотеза распределения**: Слова, которые часто встречаются в схожих контекстах, имеют одинаковое значение.

- (1) A bottle of \_\_\_\_\_ is on the table.
- (2) Everyone likes \_\_\_\_\_ .
- (3) \_\_\_\_\_ makes you drunk.
- (4) We make \_\_\_\_\_ out of corn.



	(1)	(2)	(3)	(4)	...
tezgüino	1	1	1	1	
loud	0	0	0	0	
motor oil	1	0	0	1	
tortillas	0	1	0	1	
wine	1	1	1	0	

rows are  
similar

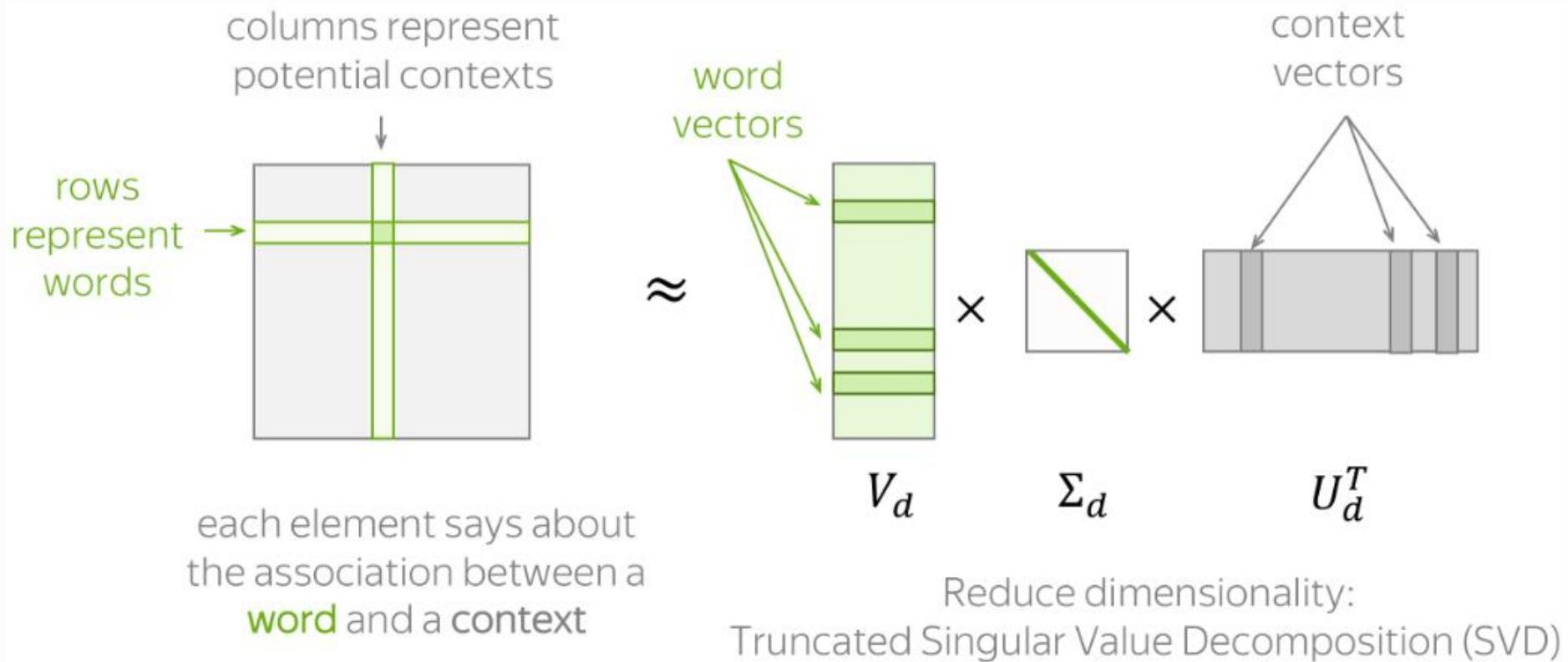


Is this true?



Согласно гипотезе распределения, «улавливать смысл» и «улавливать контексты» по своей сути одно и то же. Следовательно, все, что нужно сделать, это поместить информацию о контекстах слова в представление слова.

# Методы, основанные на подсчете



Чтобы оценить сходство между словами/контекстами, обычно необходимо оценить скалярное произведение нормализованных векторов слов/контекстов (т. е. косинусное сходство).

Основная идея : мы должны поместить информацию о контекстах в векторы слов.

Методы на основе подсчета воспринимают эту идею буквально:

Как : разместить эту информацию **вручную** на основе глобальной статистики корпуса.

Методы основанные на подсчете различаются:

- возможными контекстами,
- формулами для вычисления матричных элементов.

Самый простой подход - определить контексты каждого слова в окне L-размера. Элемент матрицы для пары слово-контекст ( $w, c$ ) - это количество раз, когда  $w$  появляется в контексте  $c$ . Это самый простой (и очень старый) метод получения вложений.



Размер контекстного окна определяется целями исследования:

- установление синтагматических связей – 1-2 слова;
- установление парадигматических связей — 5-10 слов;
- установление тематических связей — 50 слов и больше.

# Модели дистрибутивной семантики

Существует множество **различных моделей дистрибутивной семантики**, которые различаются по следующим параметрам:

- тип контекста: размер контекста, правый или левый контекст, ранжирование;
- количественная оценка частоты встречаемости слова в данном контексте: абсолютная частота, TF-IDF, энтропия, совместная информация и пр.;
- мера расстояния между векторами: косинус, скалярное произведение, расстояние Минковского и пр.;
- метод уменьшения размерности матрицы: случайная проекция, сингулярное разложение, случайное индексирование и пр.

Наиболее широко известны следующие дистрибутивно-семантические модели:

- Модель векторных пространств
- Латентно-семантический анализ
- Тематическое моделирование
- Предсказательные модели

# Word2Vec: метод, основанный на прогнозировании

Основная идея : мы должны поместить информацию о контекстах в векторы слов.

В то время как методы, основанные на подсчете, восприняли эту идею буквально, Word2Vec использует ее по-другому:

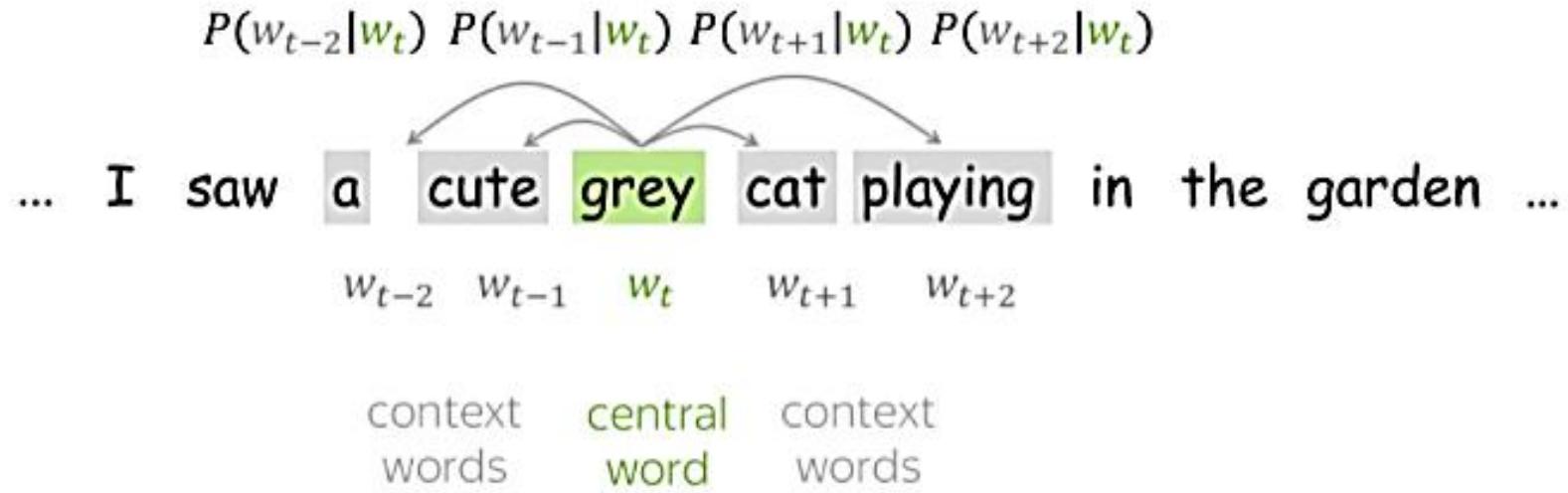
Как : Изучите векторы слов, научив их предсказывать контекст .

Word2Vec — это модель, параметрами которой являются векторы слов. Эти параметры итеративно оптимизируются для определенной цели. Цель заставляет векторы слов «знать» контексты, в которых слово может появиться: векторы обучаются предсказывать возможные контексты соответствующих слов.

Основная идея метода Word2Vec заключается в следующем:

- взять огромный текстовый корпус;
- пробежать по тексту с помощью скользящего окна, перемещая по одному слову за раз. На каждом шаге есть центральное слово и контекстные слова (другие слова в этом окне);
- для центрального слова вычислить вероятности контекстных слов;
- настроить векторы, чтобы увеличить эти вероятности.

# Word2Vec: a Prediction-Based Method

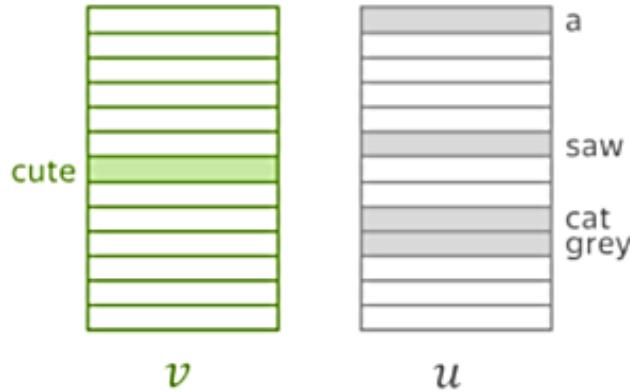


- взять огромный текстовый корпус;
- пробежать по тексту с помощью скользящего окна, перемещая по одному слову за раз. На каждом шаге есть центральное слово и контекстные слова (другие слова в этом окне);
- для центрального слова вычислить вероятности контекстных слов;
- настроить векторы, чтобы увеличить эти вероятности.



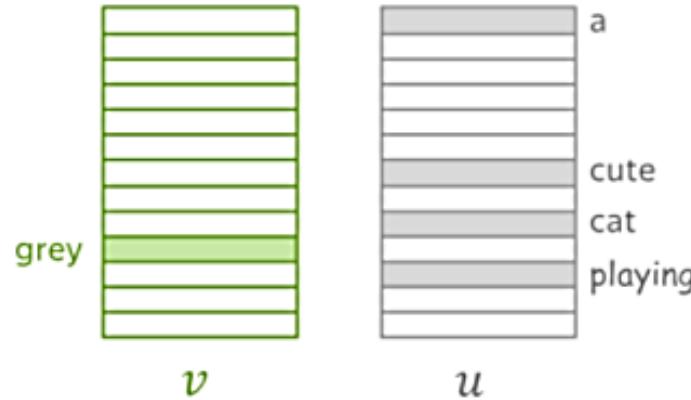
... I saw a cute grey cat playing in the garden ...

$w_{t-2} \quad w_{t-1} \quad w_t \quad w_{t+1} \quad w_{t+2}$



... I saw a cute grey cat playing in the garden ...

$w_{t-2} \quad w_{t-1} \quad w_t \quad w_{t+1} \quad w_{t+2}$



Подробнее см.

[https://lena-voita.github.io/nlp\\_course/word\\_embeddings.html#main\\_content](https://lena-voita.github.io/nlp_course/word_embeddings.html#main_content)

Семантическая близость между лингвистическими единицами вычисляется как расстояние между векторами. Чаще всего используется косинусная мера, которая вычисляется по формуле:

$$\frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

дe А и В — два вектора, расстояние между которыми вычисляется.

После проведения подобного анализа становится возможным выявить наиболее близкие по смыслу слова по отношению к изучаемому слову.

Пример наиболее близких слов к слову **КОШКА** (список получен на основании данных веб-корпуса русского языка, обработка корпуса выполнена системой Sketch Engine

Lemma	Score
<a href="#">кот</a>	0.3
<a href="#">собака</a>	0.288
<a href="#">птица</a>	0.219
<a href="#">зверь</a>	0.215
<a href="#">пес</a>	0.214
<a href="#">животное</a>	0.21
<a href="#">волк</a>	0.199
<a href="#">мальчик</a>	0.198
<a href="#">девочка</a>	0.197
<a href="#">медведь</a>	0.196
<a href="#">крыса</a>	0.186
<a href="#">парень</a>	0.174
<a href="#">мама</a>	0.172
<a href="#">корова</a>	0.168
<a href="#">папа</a>	0.164
<a href="#">лошадь</a>	0.164
<a href="#">мышь</a>	0.164

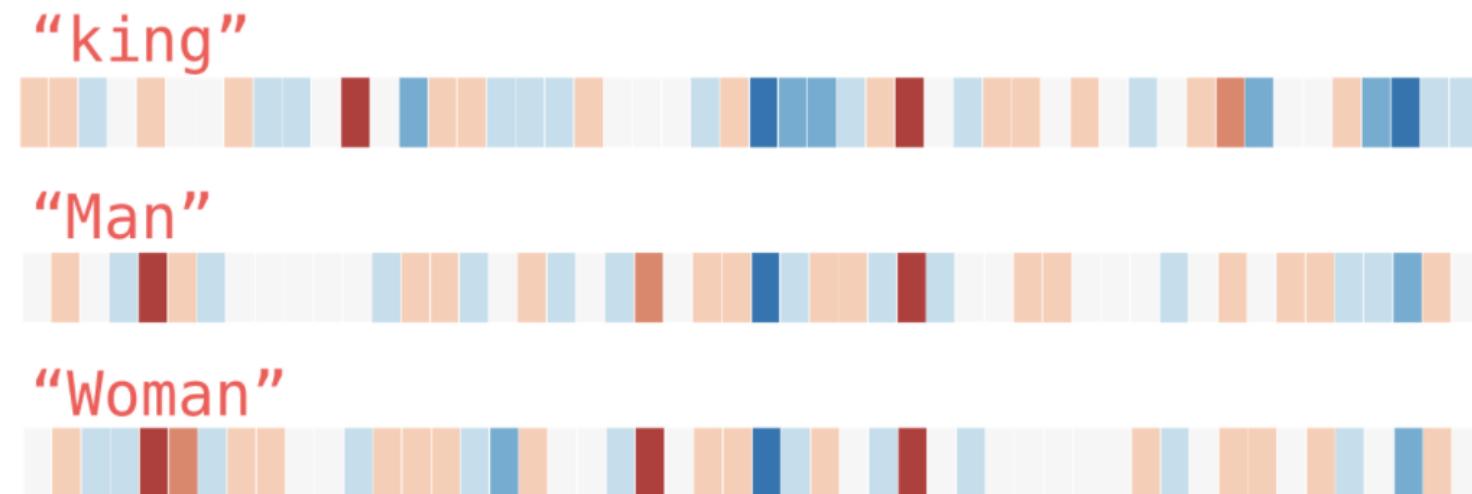


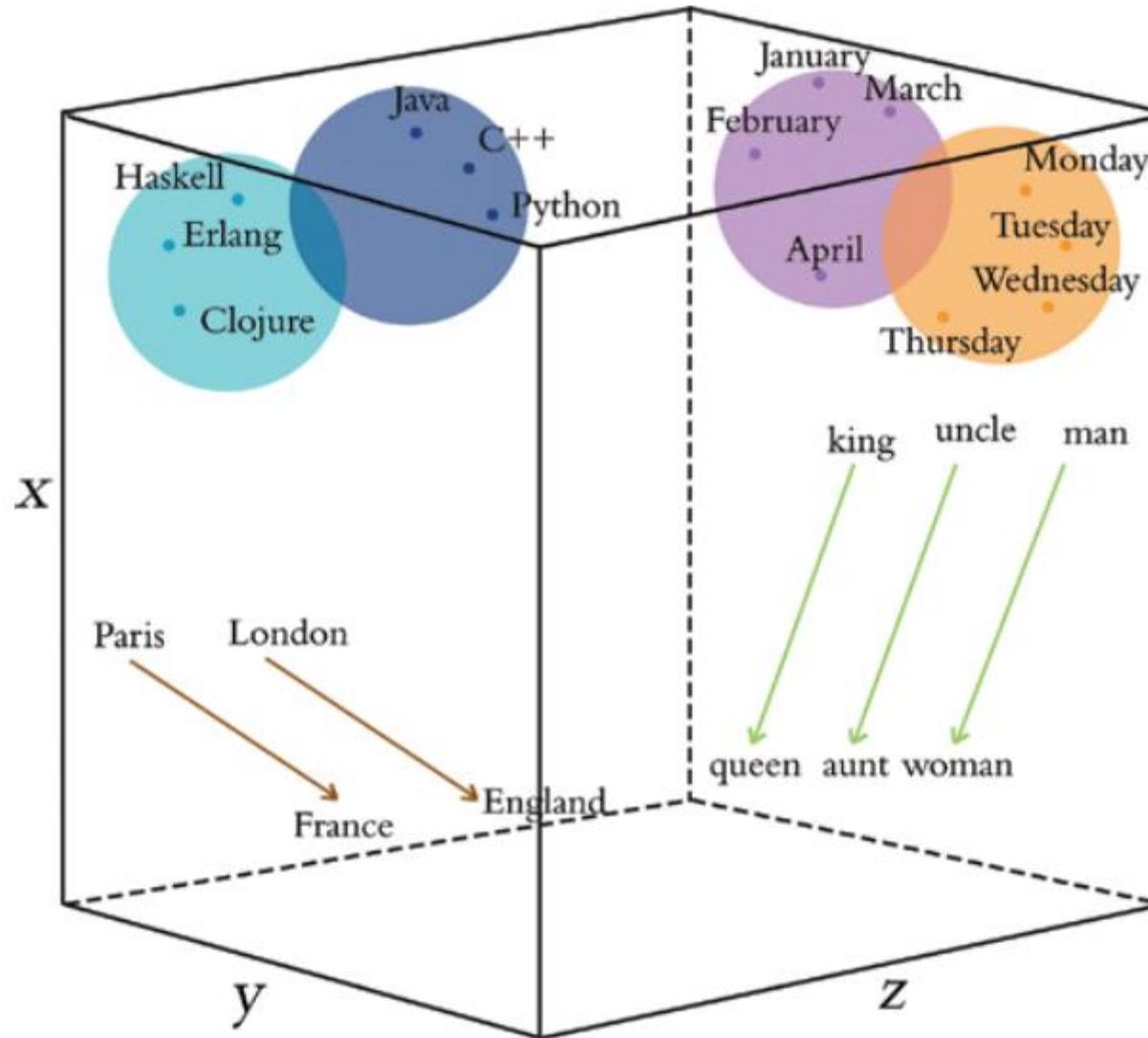
Вот вложение для слова «король» (вектор GloVe, обученный на Википедии):

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377  
, -0.61798 , -0.31012 , -0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 ,  
-0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961 , -0.13495 , -0.11476 , -0.30344 ,  
0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 , -0.04234 ,  
-0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 ,  
0.40102 , 1.1685 , -1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 ,  
-0.64426 , -0.51042 ]
```

Мы видим список из 50 чисел, но по ним трудно что-то сказать. Давайте их визуализируем, чтобы сравнить с другими векторами. Поместим числа в один ряд.:

Векторные представления слов, полученным в результате обучения называют **вложениями (embeddings)**.

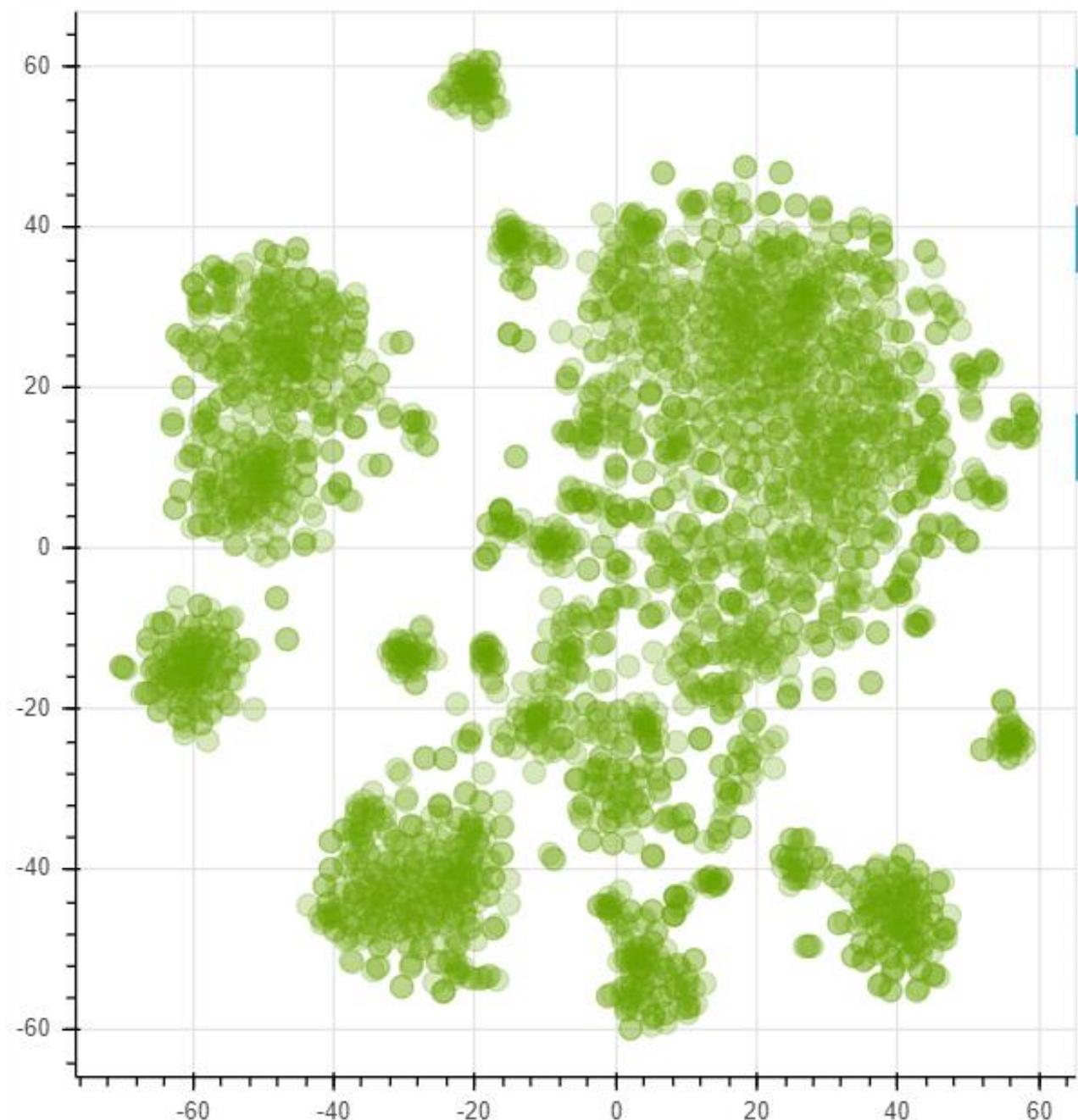




п-мерное пространство

Семантические пространства нацелены на создание представлений естественного языка, улавливающих смысл. Мы можем сказать, что (хорошие) вложения слов образуют семантическое пространство и будем называть набор векторов слов в многомерном пространстве «семантическим пространством».

На рис показано семантическое пространство, образованное векторами GloVe, обученными на данных twitter (взято из gensim). Векторы проецировались в двумерное пространство с помощью t-SNE; это только топ-3k самых частых слов.



Точки (векторы), которые находятся рядом, обычно имеют близкое значение. Иногда даже редкие слова понимаются очень хорошо.

Closest to **frog**:

**frogs**

**toad**

**litoria**

**leptodactylidae**

**rana**

**lizard**

**eleutherodactylus**

**litoria**



**leptodactylidae**



**rana**



**eleutherodactylus**



**Тесты схожести слов к ближайшим соседям** (по косинусному сходству или евклидову расстоянию) - один из методов оценки качества изученных вложений.

Существует несколько тестов (наборов тестов) на подобие слов. Они состоят из пар слов с оценкой сходства в соответствии с человеческими суждениями. Качество встраивания оценивается как корреляция между двумя оценками сходства (от модели и от людей).

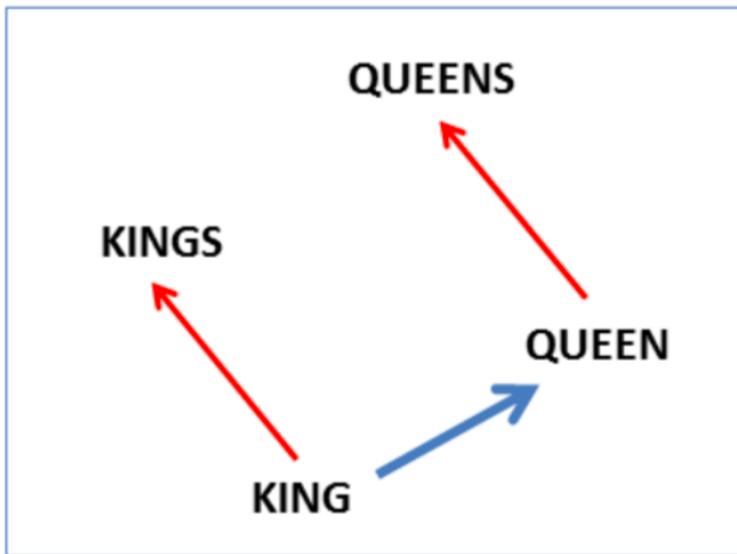
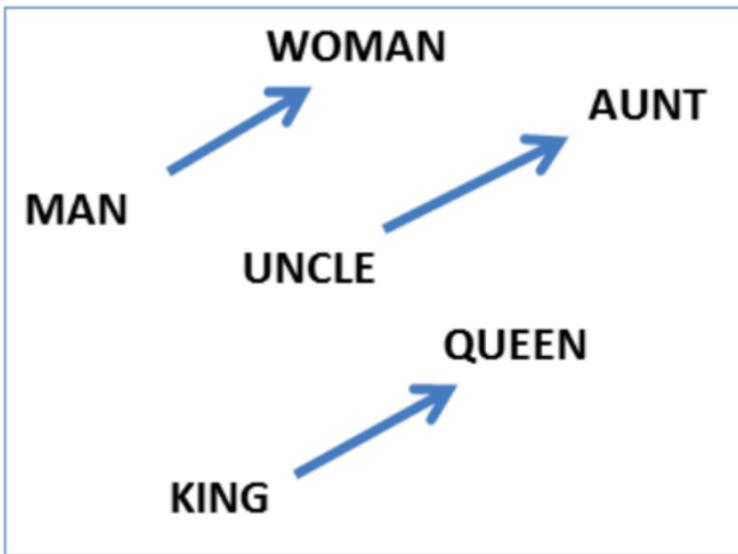
<u>word pair</u>		<u>score</u>
vulgarism	profanity	9.62
subdividing	separate	8.67
friendships	brotherhood	7.5
exceedance	probability	5.0
assigned	allow	3.5
marginalize	interact	2.5
misleading	beat	1.25
radiators	beginning	0

## Линейная структура

Удивительно то, что многие семантические и синтаксические отношения между словами (почти) линейны в векторном пространстве слов. Например, разница между королем и королевой (почти) такая же, как между мужчиной и женщиной. Или слово, которое похоже на королеву в том же смысле, что короли похожи на короля, оказывается ферзями.

semantic:  $v(\text{king}) - v(\text{man}) + v(\text{woman}) \approx v(\text{queen})$

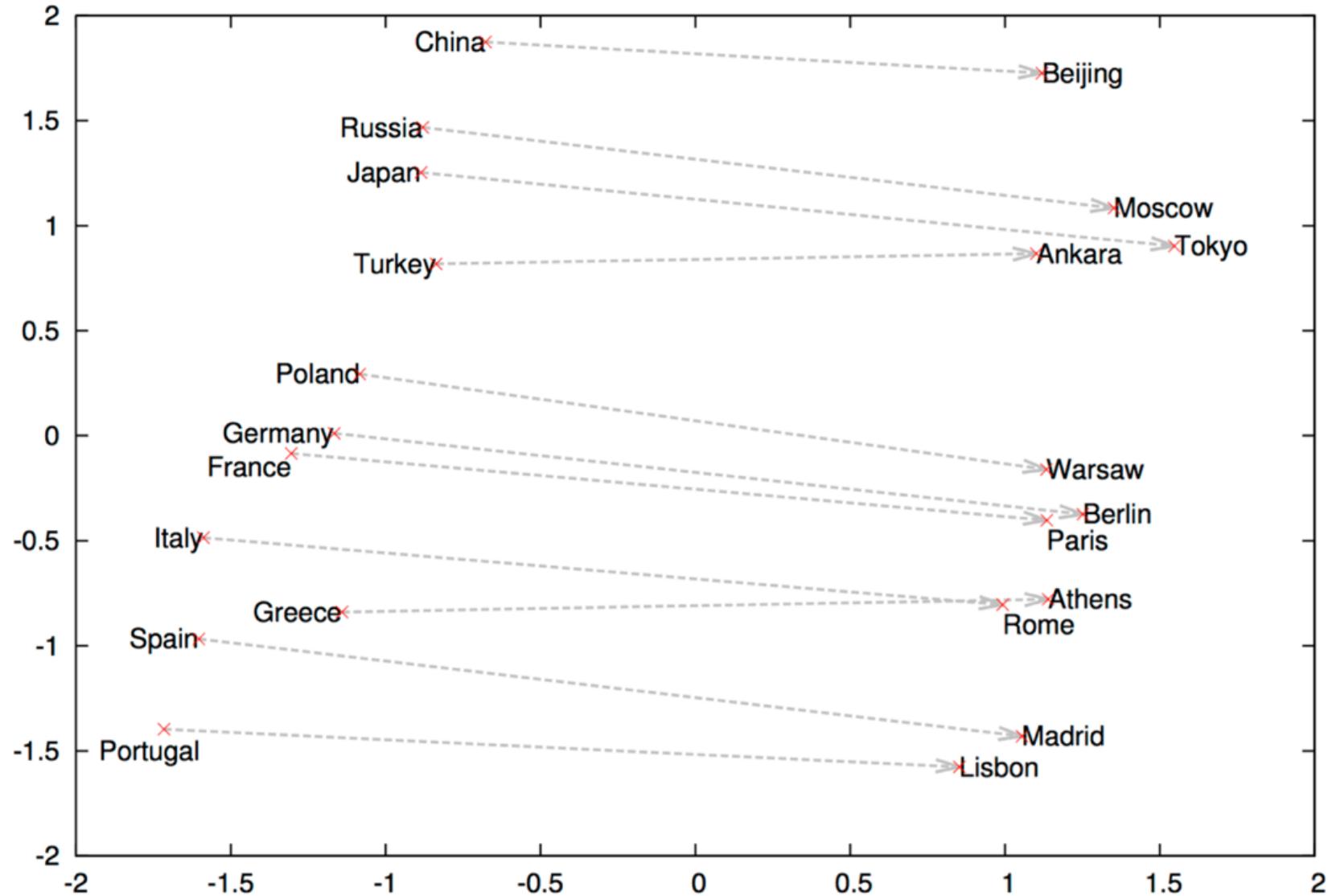
syntactic:  $v(\text{kings}) - v(\text{king}) + v(\text{queen}) \approx v(\text{queens})$



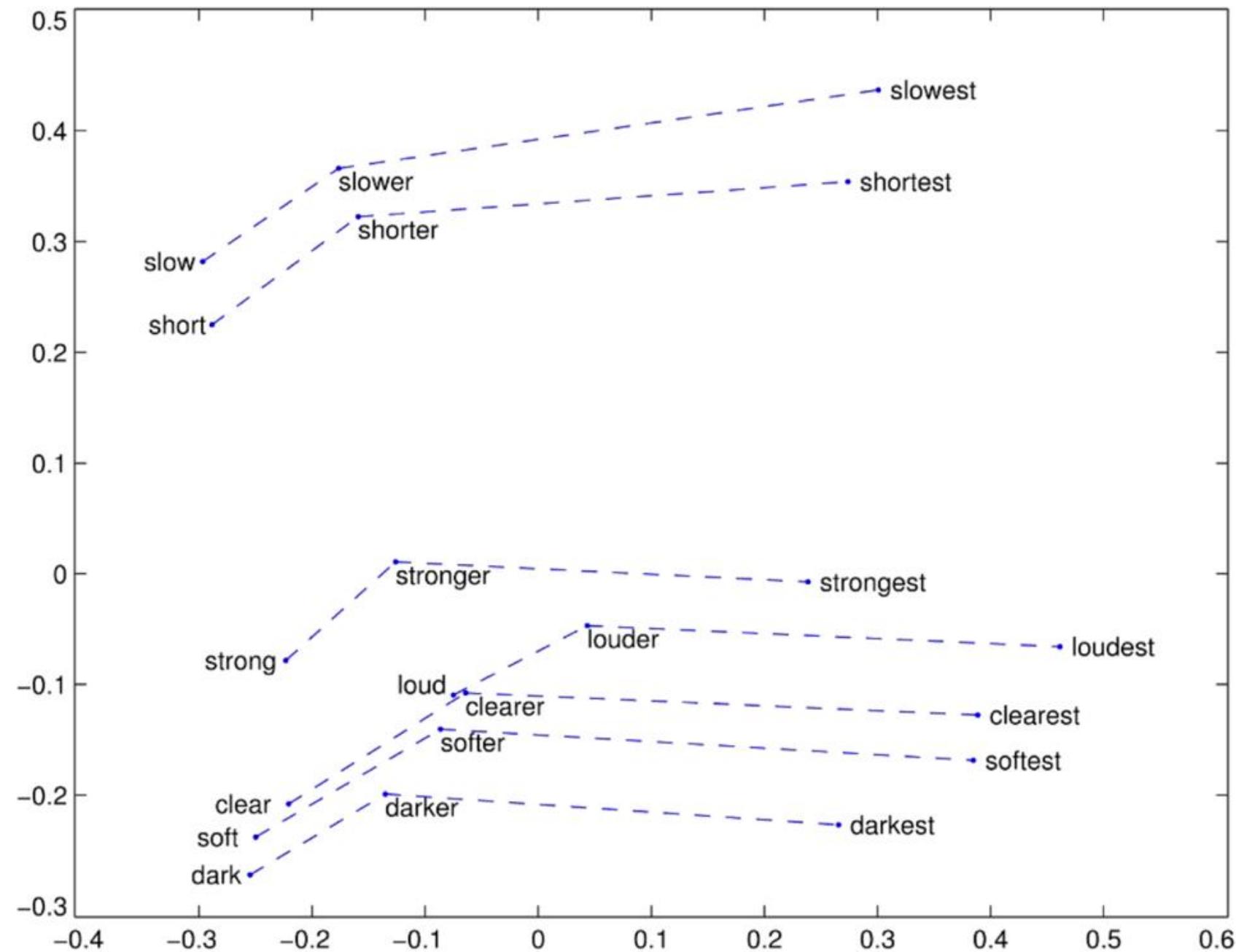
Пример мужчина-женщина ≈ король-королева самый популярный, но есть и другие интересные закономерности



### Country and Capital Vectors Projected by PCA



Word2Vec



## Сходства между языками

Отношения между семантическими пространствами разных языков также (в некоторой степени) линейны: вы можете линейно отображать одно семантическое пространство в другое, чтобы соответствующие слова в двух языках совпадали в новом совместном семантическом пространстве.

### The recipe for building large dictionaries from small ones

#### Ingredients:

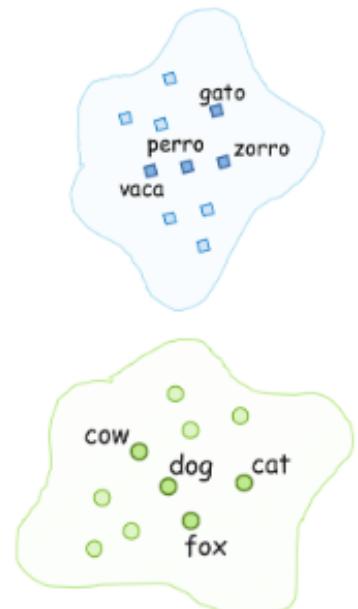
- corpus in one language (e.g., English)
- corpus in another language (e.g., Spanish)
- very small dictionary

$\text{cat} \leftrightarrow \text{gato}$   
 $\text{cow} \leftrightarrow \text{vaca}$   
 $\text{dog} \leftrightarrow \text{perro}$   
 $\text{fox} \leftrightarrow \text{zorro}$

...

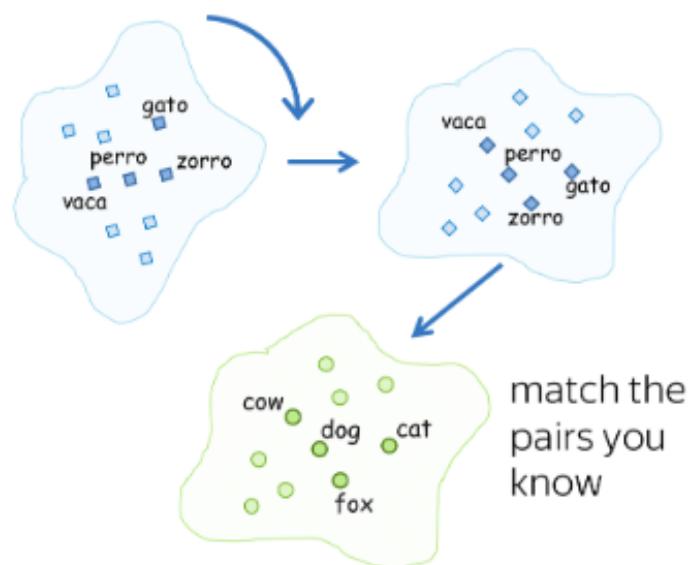
#### Step 1:

- train embeddings for each language



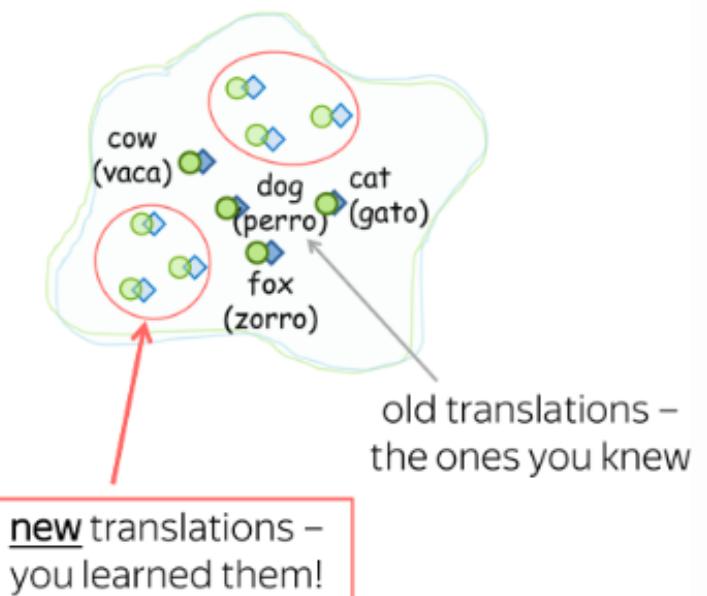
#### Step 2:

- linearly map one embeddings to the other to match words from the dictionary



#### Step 3:

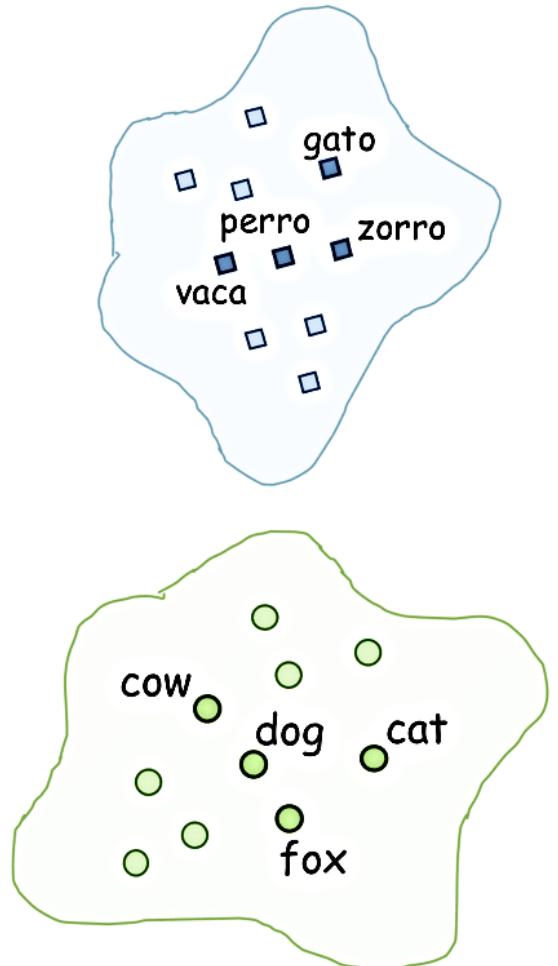
- after matching the two spaces, get new pairs from the new matches



# Перевод с иностранных языков с помощью вложения слов

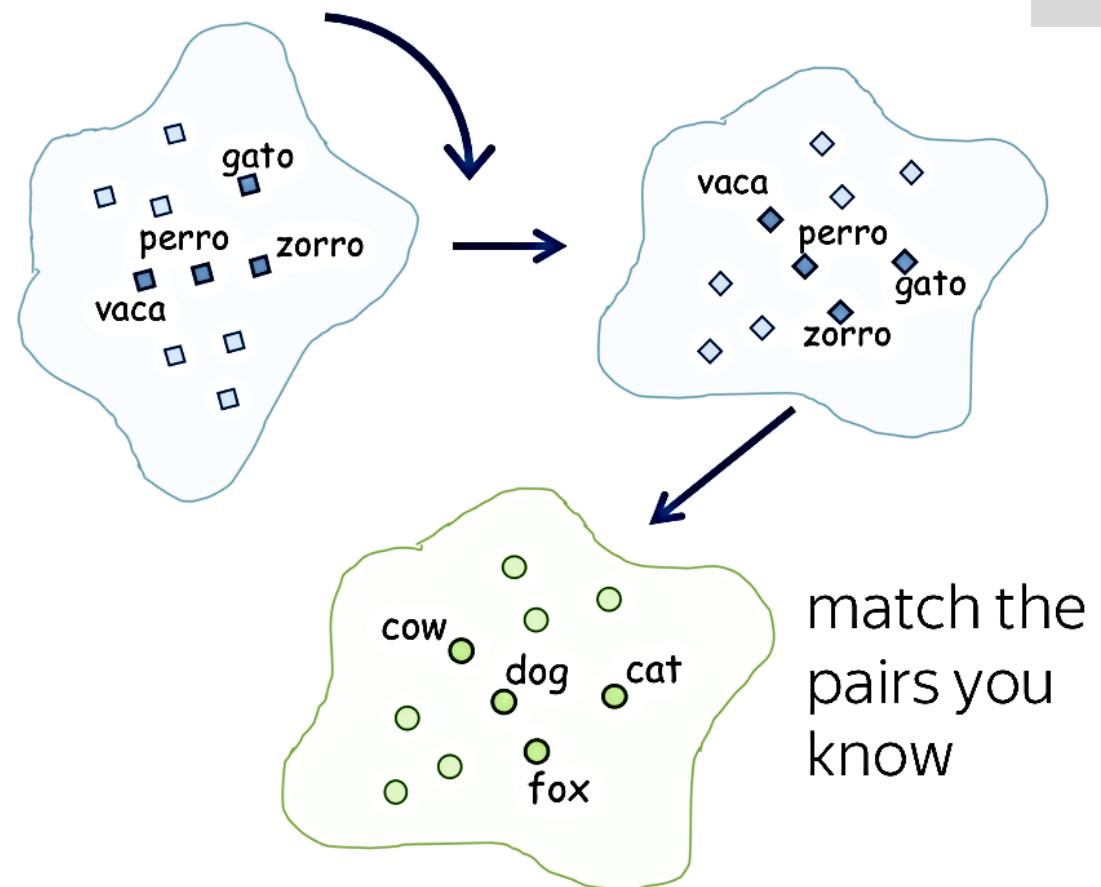
## Step 1:

- train embeddings for each language



## Step 2:

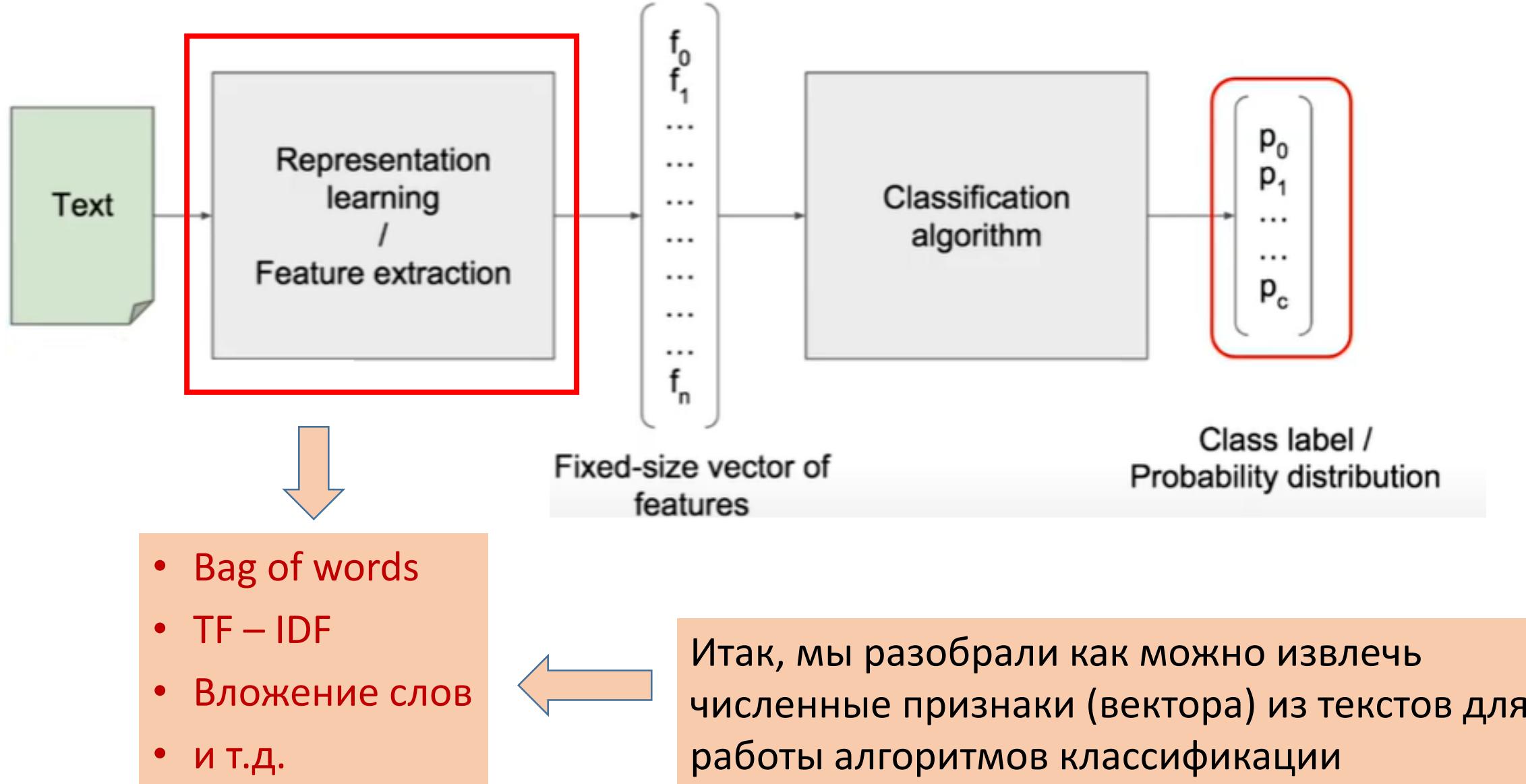
- linearly map one embeddings to the other to match words from the dictionary



**Методы word embeddings реализованы в Gensim.**

Это библиотека обработки естественного языка предназначения для «Тематического моделирования».

С ее помощью можно обрабатывать тексты, работать с векторными моделями слов (такими как Word2Vec, FastText и т. д.) и создавать тематические модели текстов.



# Примеры классификации текста



# Bag of words

```
from sklearn.feature_extraction.text import CountVectorizer

corpus = [
    'All my cats in a row',
    'When my cat sits down, she looks like a Furby toy!',
    'The cat from outer space',
    'Sunshine loves to sit like this for some reason.'
]

# Векторизация предложений при помощи "мешка слов"
vectorizer = CountVectorizer()
print( vectorizer.fit_transform(corpus).todense() )
```

! Не учитывает  
порядок слов

```
[[1 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 0 1 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1]
 [0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1 0 0]]
```

Длина вектора 26

# Bag of words

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max_features=1500, min_df=5, max_df=0.7, stop_words=)
X = vectorizer.fit_transform(documents).toarray()
```

Параметры: включаем 1500 наиболее употребляемых слов, которые встречаются минимум в 5 документах и максимум в 70% документов

# Bag of words для n-gram

```
# словарь  
vectorizer.get_feature_names_out()
```

```
array(['all', 'cat', 'cats', 'down', 'for', 'from', 'furby', 'in', 'like',  
'looks', 'loves', 'my', 'outer', 'reason', 'row', 'she', 'sit',  
'sits', 'some', 'space', 'sunshine', 'the', 'this', 'to', 'toy',  
'when'], dtype=object)
```

```
vectorizer2 = CountVectorizer(analyzer='word', ngram_range=(2, 2))  
X2 = vectorizer2.fit_transform(corpus)
```

```
# словарь n-gram n=2  
vectorizer2.get_feature_names_out()
```

```
array(['all my', 'cat from', 'cat sits', 'cats in', 'down sne',  
'for some', 'from outer', 'furby toy', 'in row', 'like furby',  
'like this', 'looks like', 'loves to', 'my cat', 'my cats',  
'outer space', 'she looks', 'sit like', 'sits down', 'some reason',  
'sunshine loves', 'the cat', 'this for', 'to sit', 'when my'],  
dtype=object)
```

Попытки учитывать  
порядок слов

```
[[1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]  
[0 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 1]  
[0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0]  
[0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 1 1 0]]
```

# Классификация текста (разбор статьи)

<https://towardsdatascience.com/text-classification-in-python-dd95d264c802>

Как правило «дается определенный набор данных (с уже назначенными ярлыками, если это проблема контролируемого обучения), мы пробуем несколько моделей и получаем метрику производительности. И на этом процесс заканчивается».

Но на деле «Как модель отреагирует на новые данные? Будут ли эти данные выглядеть так же, как набор обучающих данных? Возможно, понадобится какая-то информация (информация о масштабировании или функциях)?».

**Рассмотрим весь процесс создания службы на основе машинного обучения.**

Процесс построения модели классификации текста можно представить в виде следующих шагов:

- ✓ Определение проблемы и подход к решению
- ✓ Входные данные
- ✓ Создание начального набора данных
- ✓ Исследовательский анализ данных
- ✓ Характеристика
- ✓ Прогнозные модели

Цель работы: разработать модель, которая предсказывает категорию новостной статьи.

## 1. Постановка проблемы

Это задача классификации, относится к задачам обучения с учителем. Это означает, что нам **нужен размеченный набор данных**, чтобы алгоритмы могли изучать закономерности и корреляции в данных.

## 2. Исходные данные

Набор данных, используемый в этом проекте, - это набор необработанных данных BBC News. (<http://mlg.ucd.ie/datasets/bbc.html>). Он состоит из 2225 документов с новостного веб-сайта BBC, соответствующих сообщениям в пяти тематических областях с 2004 по 2005 годы.

- Бизнес
- Развлечение
- Политика
- Спорт
- Техника

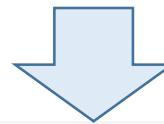


Файл загрузки содержит пять папок (по одной для каждой категории). В каждой папке есть отдельный файл .txt для каждой новостной статьи. Эти файлы включают тело новостной статьи в необработанном тексте.

### 3. Создание исходного набора данных

Из необработанных данных необходимо получить набор данных со следующей структурой:

File Name	Content	Category
Document 1 Name	Document 1 Content	Document 1 Category
Document 2 Name	Document 2 Content	Document 2 Category
...	...	...



```
df.head()
```

	File_Name	Content	Category	Complete_Filename
0	001.txt	Ad sales boost Time Warner profit\r\n\r\nQuart...	business	001.txt-business
1	002.txt	Dollar gains on Greenspan speech\r\n\r\nThe do...	business	002.txt-business
2	003.txt	Yukos unit buyer faces loan claim\r\n\r\nThe o...	business	003.txt-business
3	004.txt	High fuel prices hit BA's profits\r\n\r\nBriti...	business	004.txt-business
4	005.txt	Pernod takeover talk lifts Domecq\r\n\r\nShare...	business	005.txt-business

## 4. Исследовательский анализ данных

Одна из наших основных проблем при разработке модели классификации заключается в том, **сбалансированы ли различные классы**. Это означает, что набор данных содержит примерно равную часть каждого класса.

*Например, если бы у нас было два класса и 95% наблюдений, принадлежащих одному из них, глупый классификатор, который всегда выводит класс большинства, имел бы точность 95%, хотя он не смог бы выполнить все прогнозы класса меньшинства.*

Нередко возникают ситуации, когда в обучающем наборе данных **доля примеров некоторого класса слишком мала** (этот класс будем называть миноритарным, а другой, сильно представленный, — мажоритарным). Такие тенденции хорошо заметны в кредитном scoringе, в медицине, в direct-mail маркетинге. **Построенный на таких наборах данных классификатор может оказаться абсолютно неэффективным.**

Восстановление баланса классов может проходить двумя путями. В первом случае удаляют некоторое количество примеров мажоритарного класса (*undersampling*), во втором — увеличивают количество примеров миноритарного (*oversampling*).

# Количества статей по категориям

<https://github.com/miguelzafra/Latest-News-Classifier/blob/master/0.%20Latest%20News%20Classifier/02.%20Exploratory%20Data%20Analysis/02.%20Exploratory%20Data%20Analysis.ipynb>

```
bars = alt.Chart(df).mark_bar(size=50).encode(  
    x=alt.X("Category"),  
    y=alt.Y("count():Q", axis=None),  
    tooltip=[alt.Tooltip('count():Q',  
        color='Category')]  
)  
  
text = bars.mark_text(  
    align='center',  
    baseline='bottom',  
).encode(  
    text='count()'  
)  
  
(bars + text).interactive()  
    height=300,  
    width=700,  
    title = "Number of articles"  
)
```



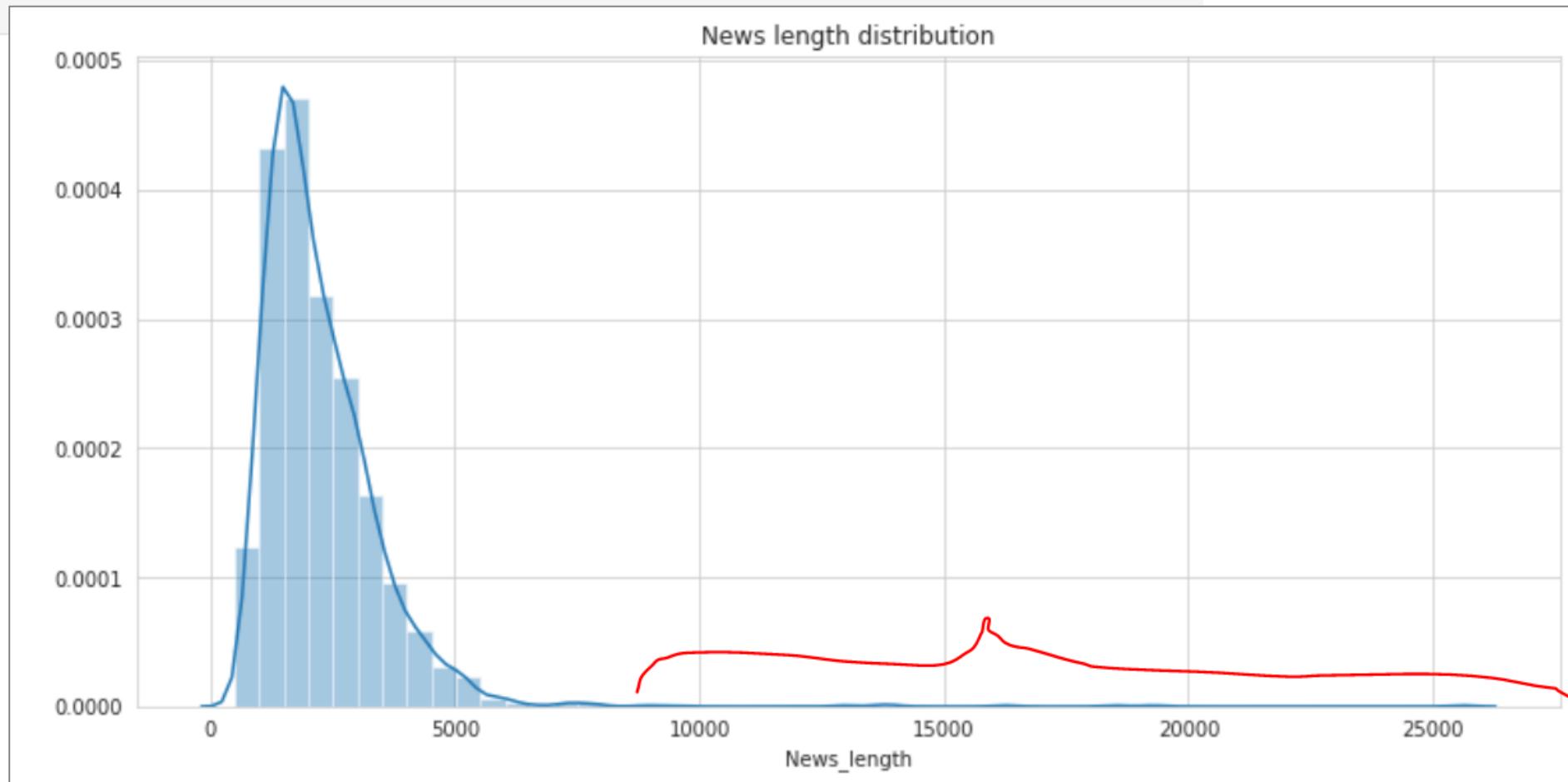
Строим гистограмму для оценки количества статей по категориям

Другой интересной переменной может быть длина новостных статей. Мы можем получить **распределение длины по категориям**:

```
df['News_length'] = df['Content'].str.len()
```

News length by category

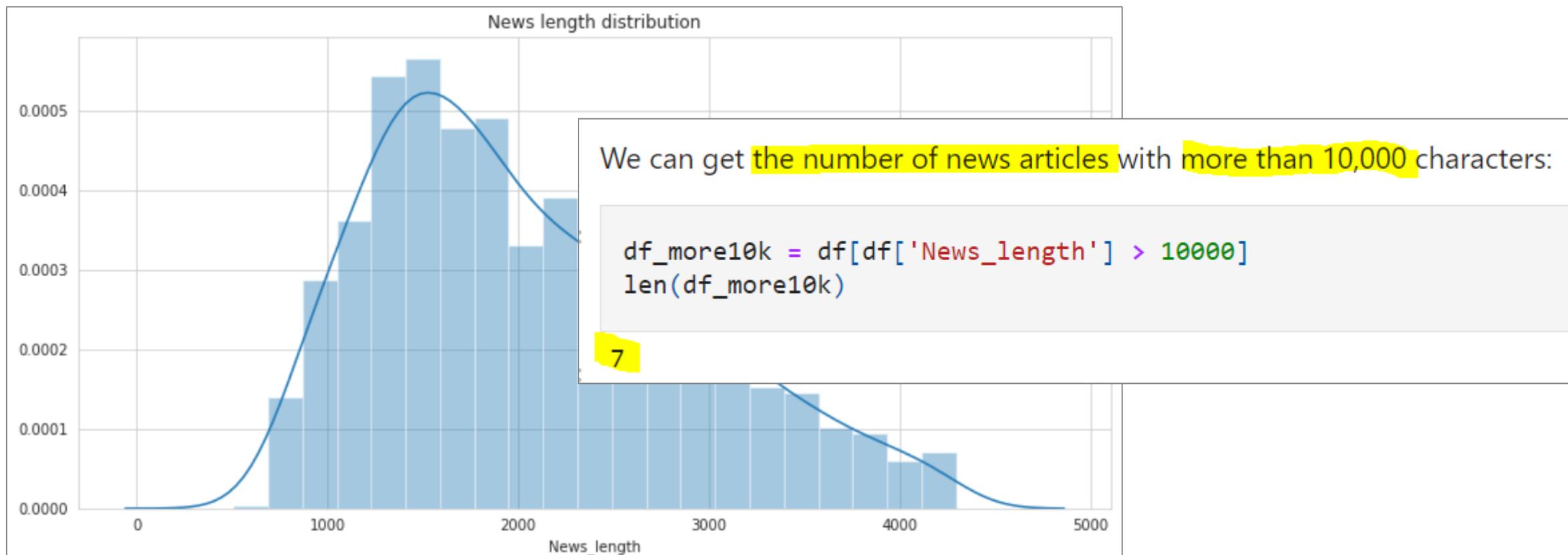
```
plt.figure(figsize=(12.8,6))
sns.distplot(df['News_length']).set_title('News length distribution');
```



Let's remove from the 95% percentile onwards to better appreciate the histogram:

```
quantile_95 = df['News_length'].quantile(0.95)
df_95 = df[df['News_length'] < quantile_95]
```

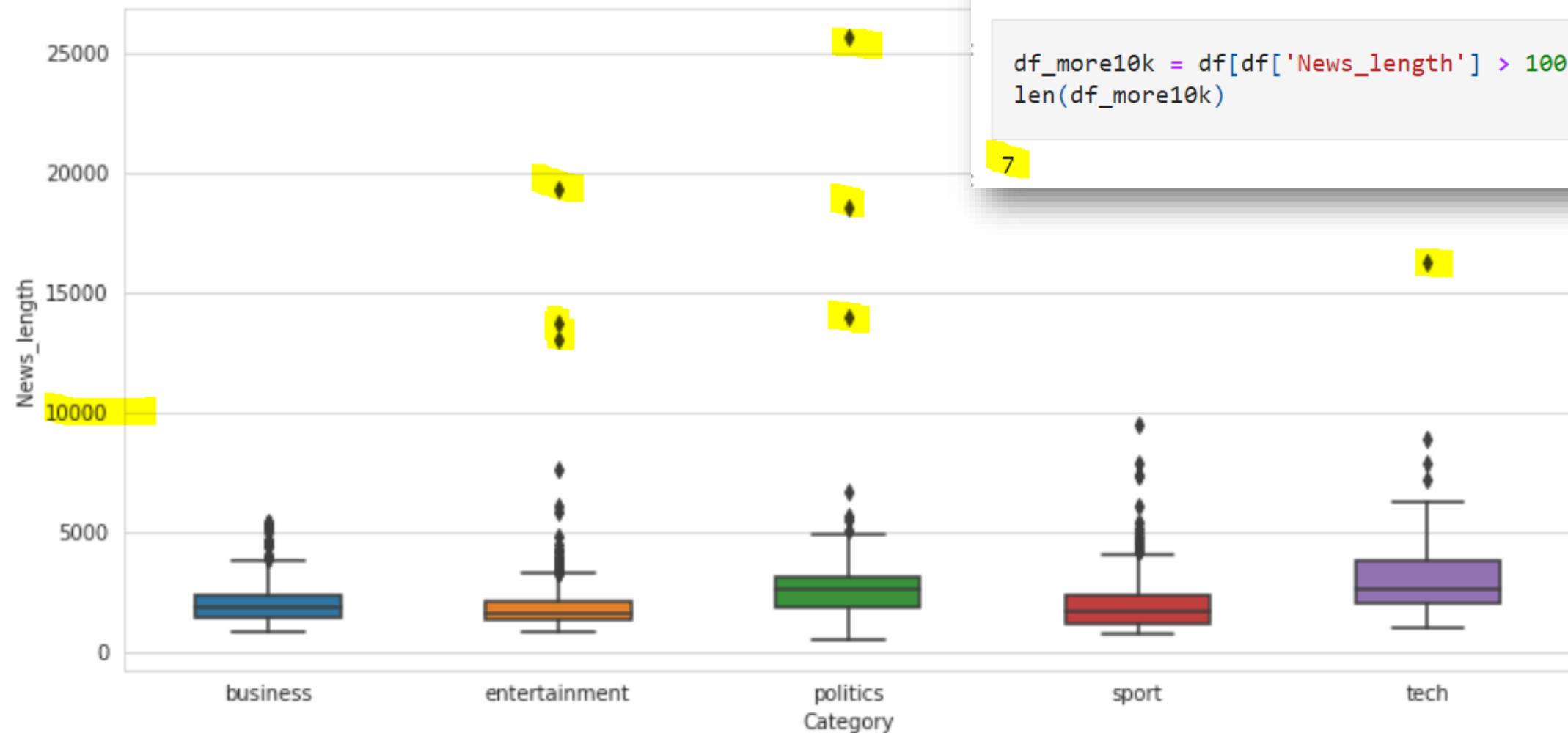
```
plt.figure(figsize=(12.8,6))
sns.distplot(df_95['News_length']).set_title('News length distribution');
```



Let's now plot a boxplot:

# Boxplot

```
plt.figure(figsize=(12.8,6))
sns.boxplot(data=df, x='Category', y='News_length', width=.5);
```

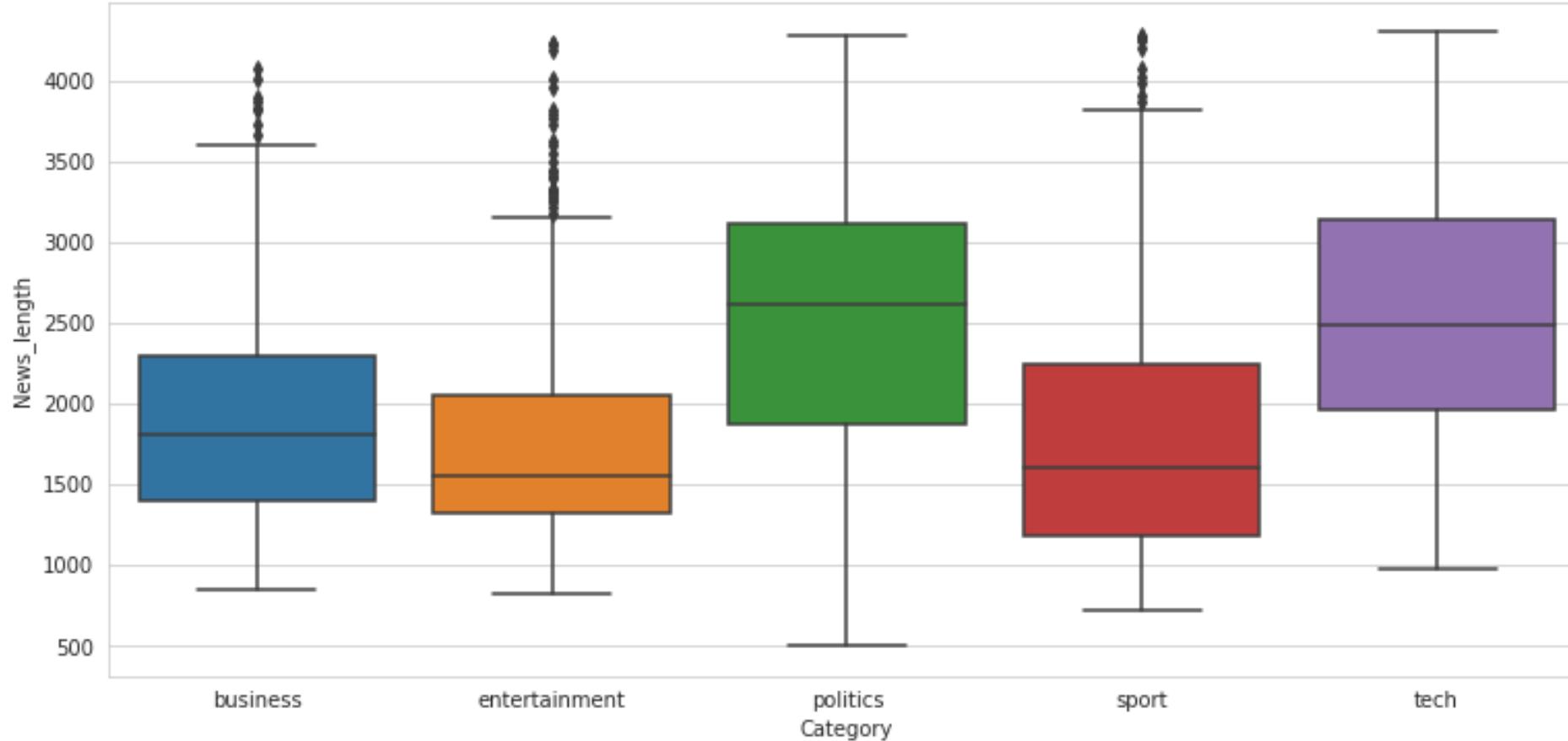


We can get the number of news articles with more than 10,000 ch

```
df_more10k = df[df['News_length'] > 10000]
len(df_more10k)
```

# Анализ длины статей по категориям с помощью boxplot

```
plt.figure(figsize=(12.8,6))
sns.boxplot(data=df_95, x='Category', y='News_length');
```



Статьи о политике и технологиях имеют тенденцию быть длиннее, но не в значительной степени.

## 5. Очистка текста

- **Удаляем специальные символы** : специальные символы, такие как двойные кавычки « \ n », должны быть удалены из текста, поскольку мы не ожидаем от них какой-либо предсказательной силы.
- **Верхний регистр / нижний регистр**: «Книга» и «книга» будут одним и тем же словом и будут иметь одинаковую предсказательную силу.
- **Знаки препинания**: такие символы, как «?», «!», «;» удаляем.
- **Притяжательные местоимения**: можно ожидать, что «Tramp» и «Tramp's» обладают одинаковой предсказательной силой.
- **Стемминг или лемматизация**: стемминг - это процесс сокращения производных слов до их корня. Лемматизация - это процесс сведения слова к его лемме. Основное различие между обоими методами заключается в том, что лемматизация предоставляет существующие слова, тогда как выделение корня дает корень, который может не быть существующим словом. Мы использовали лемматизатор на основе WordNet.
- **Стоп-слова**: такие слова, как «что» «или» не будут иметь никакой предсказательной силы, поскольку они предположительно будут общими для всех документов. По этой причине они могут представлять собой шум, который можно устраниить.

## 5. Очистка текста

1

```
# \r and \n
df['Content_Parsed_1'] = df['Content'].str.replace("\r", " ")
df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace("\n", " ")
df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace("  ", " ")
```

2

```
# Lowercasing the text
df['Content_Parsed_2'] = df['Content_Parsed_1'].str.lower()
```

3

```
punctuation_signs = list("?!:.,;")
df['Content_Parsed_3'] = df['Content_Parsed_2']

for punct_sign in punctuation_signs:
    df['Content_Parsed_3'] = df['Content_Parsed_3'].str.replace(punct_sign, '')
```

4

```
df['Content_Parsed_4'] = df['Content_Parsed_3'].str.replace("'s", "")
```

## 6. Кодирование меток

Для прогнозирования моделей машинного обучения требуются числовые функции и метки. Нужен словарь, чтобы сопоставить каждую метку с числовым идентификатором:

```
category_codes = {  
    'business': 0,  
    'entertainment': 1,  
    'politics': 2,  
    'sport': 3,  
    'tech': 4  
}
```

```
# Category mapping  
df['Category_Code'] = df['Category']  
df = df.replace({'Category_Code':category_codes})
```

## 7. Разработка функций (Векторизация)

**Разработка функций** - это процесс преобразования данных в функции, которые служат входными данными для моделей машинного обучения.

При работе с текстовыми данными есть несколько способов получения функций, представляющих данные.

### 7.1. Текстовое представление

Напомним, что для представления нашего текста каждая строка набора данных будет одним документом корпуса (набора). Столбцы (функции) будут разными в зависимости от того, какой метод создания функции мы выберем:

- Векторы количества слов (Мешок слов)
  - Векторы TF – IDF
  - Вложения слов
- Эти 2 метода не учитывают порядок слов

Положение слова в векторном пространстве изучается из текста и основывается на словах, которые окружают слово, когда оно используется.

- Текстовые или НЛП-функции

Мы можем вручную создать любую функцию, которая, по нашему мнению, может иметь значение при различении категорий (например, плотность слова, количество символов или слов и т. д.).

- Тематические модели

Такие методы, как скрытое распределение Дирихле, пытаются представить каждую тему с помощью вероятностного распределения по словам, что называется «моделирование темы».

**Для векторизации выбрали векторы TF-IDF.** Эти выборы мотивированы следующими пунктами:

- TF-IDF - это **простая модель**, которая может давать **хорошие результаты**.
- Создание функций TF-IDF - это **быстрый процесс** (который, например, сокращает время ожидания пользователя при использовании веб-приложения).
- **Можно настроить процесс создания векторов**, чтобы избежать таких проблем, как переобучение.

**При создании объектов с помощью этого метода мы можем выбрать некоторые параметры:**

- **N-граммы**: мы можем рассматривать униграммы, биграммы, триграммы...
- **Максимальная/минимальная частота**: при построении словаря мы можем игнорировать слова, у которых частота появления строго выше/ниже заданного порога.
- **Максимальное количество функций**: мы можем выбрать первые N функций, упорядоченных по частоте использования слов в корпусе.

Parameter	Value
N-gram range	(1, 2)
Maximum DF	1
Minimum DF	10
Maximum features	300

- биграммы помогут улучшить производительность модели за счет учета слов, которые обычно встречаются в документах вместе
- Minimum DF = 10, чтобы избавиться от чрезвычайно редких слов, которые встречаются не более чем в 10 документах
- Maximum DF = 100%, чтобы не игнорировать любые другие слова

```
ngram_range = (1, 2)
min_df = 10
max_df = 1.
max_features = 300
```

TfidfVectorizer преобразует набор необработанных документов в матрицу функций TF-IDF.

```
tfidf = TfidfVectorizer(encoding='utf-8',
                        ngram_range=ngram_range,
                        stop_words=None,
                        lowercase=False,
                        max_df=max_df,
                        min_df=min_df,
                        max_features=max_features,
                        norm='l2',
                        sublinear_tf=True)

features_train = tfidf.fit_transform(X_train).toarray()
labels_train = y_train
print(features_train.shape)

features_test = tfidf.transform(X_test).toarray()
labels_test = y_test
print(features_test.shape)
```

```
(1891, 300)
(334, 300)
```

## 8. Предсказательные модели

Использовали **только классические модели машинного обучения** вместо моделей глубокого обучения из-за недостаточного количества данных, которые у нас есть.

Протестированы следующие модели:

- Случайный лес
- Машина опорных векторов
- К Ближайшие соседи
- Полиномиальный наивный байесовский
- Полиномиальная логистическая регрессия
- Повышение градиента

### 8.1. Настройка гиперпараметров

- решили, какие гиперпараметры мы хотим настроить для каждой модели
- определили сетку возможных значений
- выполнили поиск по сетке

## 9.2. Измерение точности

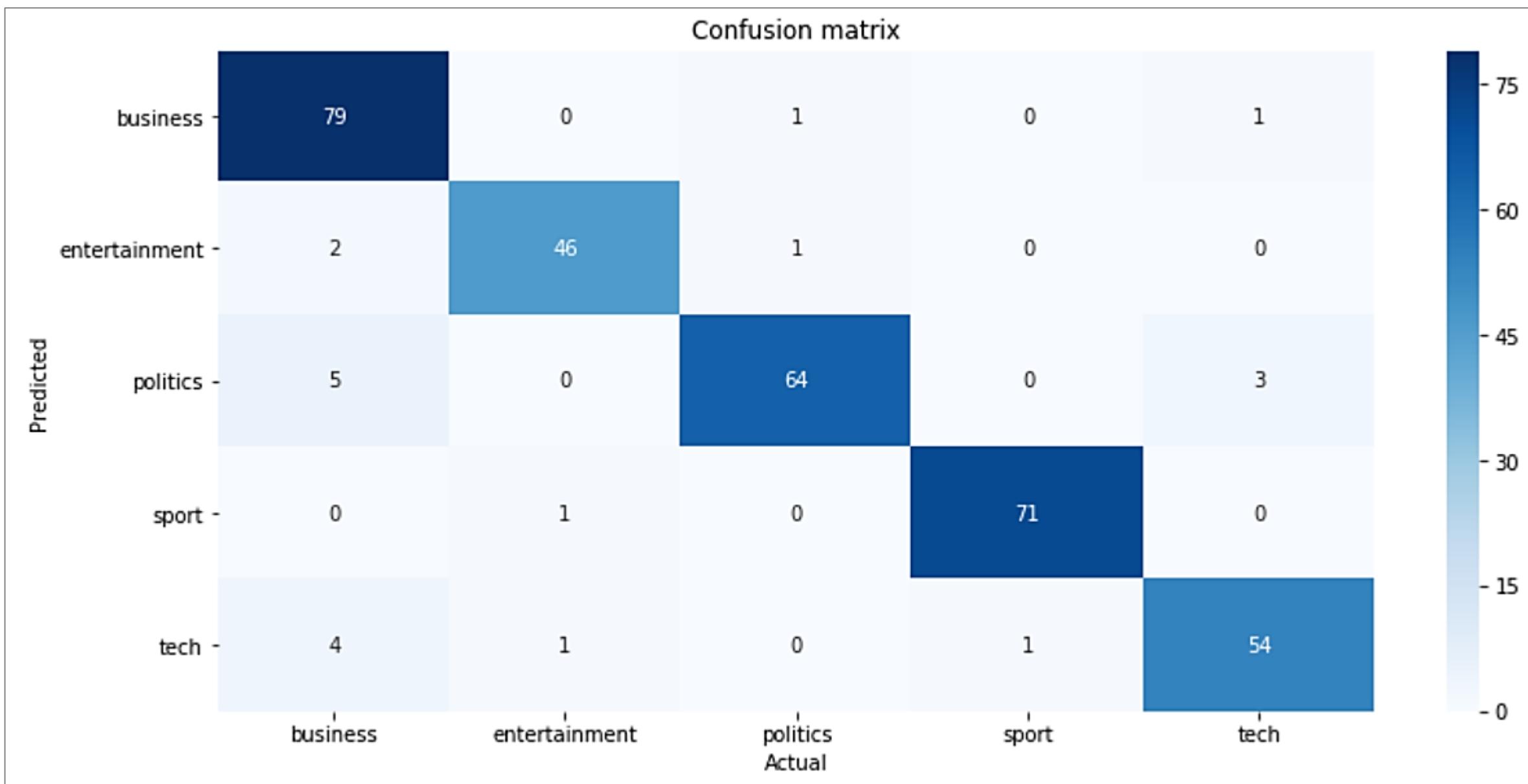
	Model	Training Set Accuracy	Test Set Accuracy
0	Gradient Boosting	1.000000	0.940120
1	KNN	0.959810	0.928144
2	Logistic Regression	0.981491	0.940120
3	Multinomial Naïve Bayes	0.953993	0.934132
4	Random Forest	1.000000	0.928144
5	SVM	0.959281	0.940120

В целом, получены хорошие значения точности для каждой модели.

Можно заметить, что модели **Gradient Boosting**, **Logistic Regression** и **Random Forest** **кажутся избыточными**, поскольку они имеют чрезвычайно высокую точность обучающего набора, но более низкую точность набора тестов, **поэтому отбросим их**.

Мы выберем классификатор SVM, потому что он имеет наивысшую точность тестового набора, что действительно близко к точности обучающего набора.

## Матрица неточностей модели SVM:



# Вступление (разбор статьи)

<https://pythobYTE.com/text-classification-with-python-and-scikit-learn-7bd1af87/>

Классификация текстов-одна из важнейших задач обработки естественного языка . Это процесс классификации текстовых строк или документов по различным категориям в зависимости от содержания строк. Классификация текста имеет множество приложений, таких как определение настроений пользователей по твиту, классификация электронной почты как спама или ветчины, классификация сообщений в блогах по различным категориям, автоматическая маркировка запросов клиентов и т. Д.

В этой статье мы рассмотрим реальный пример классификации текста. Мы будем обучать модель машинного обучения, способную предсказать, является ли данный обзор фильма положительным или отрицательным. Это классический пример сентиментального анализа, когда чувства людей по отношению к тому или иному субъекту классифицируются по различным категориям.

# Набор данных

Набор данных, который мы собираемся использовать для этой статьи, можно загрузить из [Cornell Natural Language Processing Group](#). Набор данных состоит в общей сложности из 2000 документов. Половина документов содержит положительные отзывы о фильме, а остальная половина-отрицательные. Более подробную информацию о наборе данных можно найти по ссылке [this link](#).

Распакуйте или извлеките набор данных после его загрузки. Откройте папку “txt\_send token”. Папка содержит две подпапки: “neg” и “pos”. Если вы откроете эти папки, то увидите текстовые документы, содержащие обзоры фильмов.

Ниже приведены шаги, необходимые для создания модели классификации текста в Python:

1. Импорт библиотек
2. Импорт набора данных
3. Предварительная обработка текста
4. Преобразование текста в числа
5. Учебные и тестовые наборы
6. Обучающая Модель Классификации текста и прогнозирования настроений
7. Оценка Модели
8. Сохранение и загрузка модели

# Импорт набора данных

Мы будем использовать функцию `load_files` из библиотеки `sklearn_datasets` для импорта набора данных в наше приложение. Функция `load_files` автоматически делит набор данных на наборы данных и целевые наборы. Например, в нашем случае мы передадим ему путь к каталогу “text\_send token”. Файл `load_files` будет рассматривать каждую папку внутри папки “text\_send token” как одну категорию, и всем документам внутри этой папки будет присвоена соответствующая категория.

Выполните следующий скрипт, чтобы увидеть `load_files` функцию в действии:

```
movie_data = load_files(r"D:\txt_sentoken")
X, y = movie_data.data, movie_data.target
```

```
documents = []

from nltk.stem import WordNetLemmatizer

stemmer = WordNetLemmatizer()

for sen in range(0, len(X)):
    # Remove all the special characters
    document = re.sub(r'\W', ' ', str(X[sen]))

    # remove all single characters
    document = re.sub(r'\s+[a-zA-Z]\s+', ' ', docu

    # Remove single characters from the start
    document = re.sub(r'^[a-zA-Z]\s+', ' ', docum

    # Substituting multiple spaces with single space
    document = re.sub(r'\s+', ' ', document, flags

    # Removing prefixed 'b'
    document = re.sub(r'^b\s+', '', document)

    # Converting to Lowercase
    document = document.lower()
```

## Предварительная обработка текста

В приведенном выше сценарии мы используем **регулярные выражения из библиотеки Python re** для выполнения различных задач предварительной обработки.

# Преобразование текста в числа

Машины, в отличие от людей, не могут понять сырой текст. Машины могут видеть только цифры. В частности, статистические методы, такие как машинное обучение, могут работать только с числами. Поэтому нам нужно преобразовать наш текст в числа.

Существуют различные подходы к преобразованию текста в соответствующую числовую форму. [Модель мешка слов](#) и [Модель встраивания слов](#) являются двумя наиболее часто используемыми подходами. В этой статье мы будем использовать модель мешка слов для преобразования нашего текста в числа.

## Наборы для обучения и тестирования

Как и любая другая контролируемая задача машинного обучения, мы должны разделить наши данные на обучающие и тестовые наборы. Для этого мы воспользуемся утилитой `train_test_split` из библиотеки `sklearn.model_selection`. Выполните следующий сценарий:

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split
```

Для обучения нашей модели машинного обучения с использованием алгоритма случайного леса мы будем использовать Классификатор случайного леса класс из библиотеки `sklearn.ensemble`. Метод `fit` этого класса используется для обучения алгоритма. Нам нужно передать обучающие данные и обучающие целевые наборы этому методу. Взгляните на следующий сценарий:

```
classifier = RandomForestClassifier(n_estimators=1
classifier.fit(X_train, y_train)
```

Наконец, чтобы предсказать настроение документов в нашем тестовом наборе, мы можем использовать метод `predict` класса `RandomForestClassifier`, как показано ниже:

```
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import classification_report,  
  
print(confusion_matrix(y_test,y_pred))  
print(classification_report(y_test,y_pred))  
print(accuracy_score(y_test, y_pred))
```

Вывод выглядит следующим образом:

```
[[180  28]  
 [ 30 162]]  
          precision    recall   f1-score   support  
  
          0           0.86      0.87      0.86      208  
          1           0.85      0.84      0.85      182  
  
avg / total       0.85      0.85      0.85      400  
  
0.855
```

## Сохранение и загрузка модели

Мы можем сохранить нашу модель как объект `pickle` в Python. Для этого выполните следующий сценарий:

```
with open('text_classifier', 'wb') as picklefile:  
    pickle.dump(classifier,picklefile)
```

После выполнения описанного выше скрипта вы можете увидеть файл `text_classifier` в своем рабочем каталоге. Мы сохранили нашу обученную модель и можем использовать ее позже для прямого предсказания, без обучения.

Для загрузки модели мы можем использовать следующий код:

```
with open('text_classifier', 'rb') as training_model
    model = pickle.load(training_model)
```

```
y_pred2 = model.predict(X_test)

print(confusion_matrix(y_test, y_pred2))
print(classification_report(y_test, y_pred2))
print(accuracy_score(y_test, y_pred2))
```

```
[[180  28]
 [ 30 162]]

           precision    recall  f1-score   support
          0          0.86      0.87      0.86      208
          1          0.85      0.84      0.85      172

avg / total          0.85      0.85      0.85      380

0.855
```

Результат похож на тот, который мы получили ранее,

При выполнении лабораторной работы для парсинга текстов или отзывов удобно использовать библиотеку **Beautiful Soup**

## Документация Beautiful Soup

Beautiful Soup — это библиотека Python для извлечения данных из файлов HTML и XML. Она работает с вашим любимым парсером, чтобы дать вам естественные способы навигации, поиска и изменения дерева разбора. Она обычно экономит программистам часы и дни работы.

Эти инструкции иллюстрируют все основные функции Beautiful Soup 4 на примерах. Я покажу вам, для чего нужна библиотека, как она работает, как ее использовать, как заставить ее делать то, что вы хотите, и что нужно делать, когда она не оправдывает ваши ожидания.

Примеры в этой документации работают одинаково на Python 2.7 и Python 3.2.

