



# Обработка естественного языка

Парполов Дмитрий

# Содержание

---

- Предобработка
- Признаковое описание текста
- word2vec, fasttext



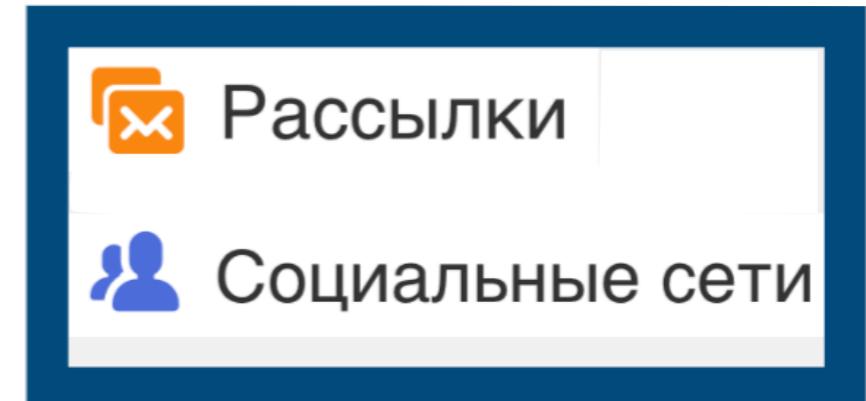
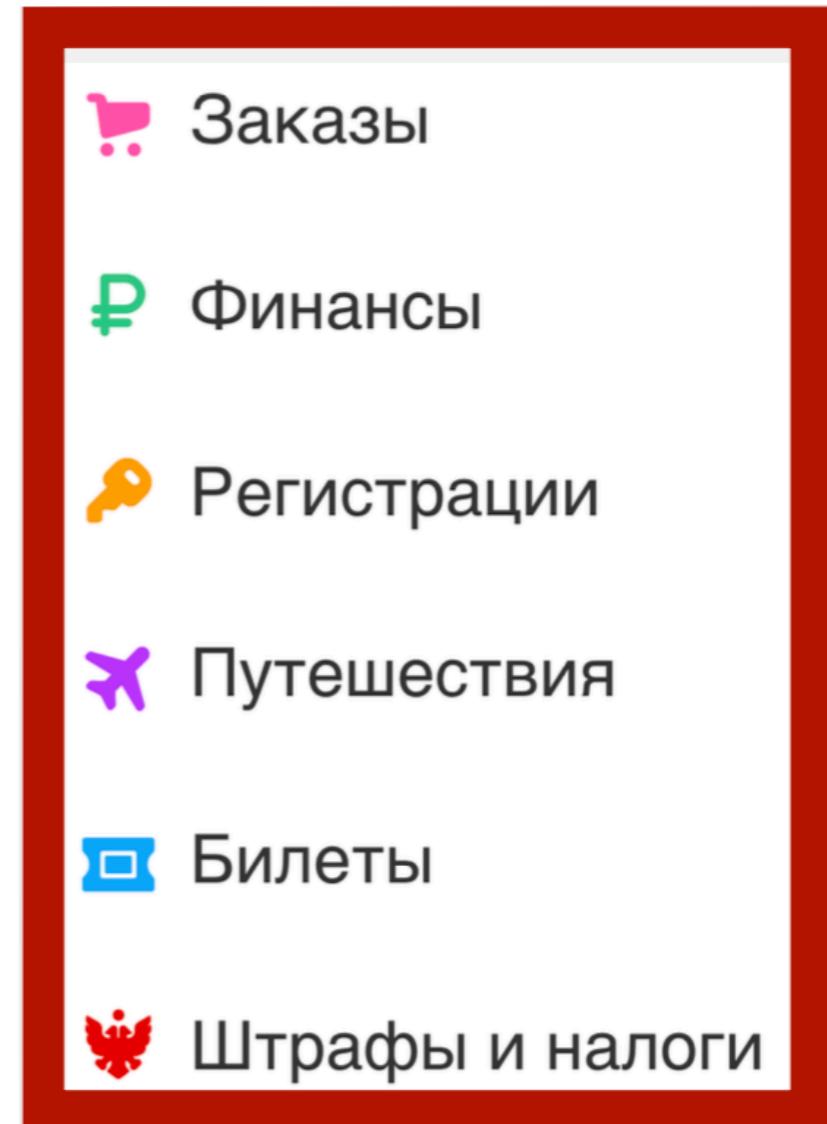
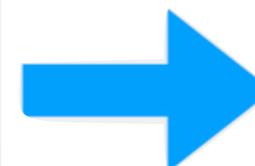
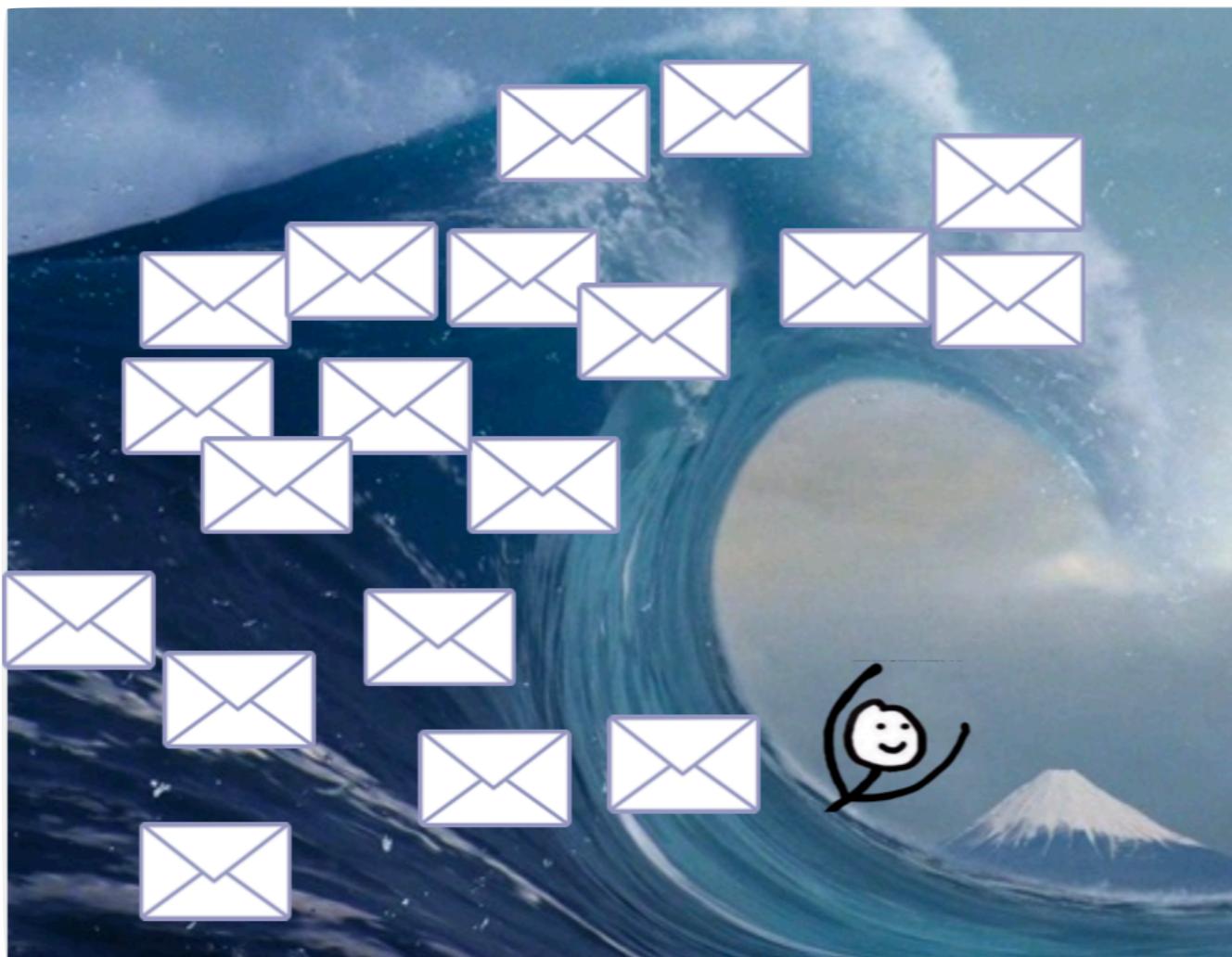


# NLP в Почте (и не только)

# Антиспам



# Категоризация



# Smart Reply



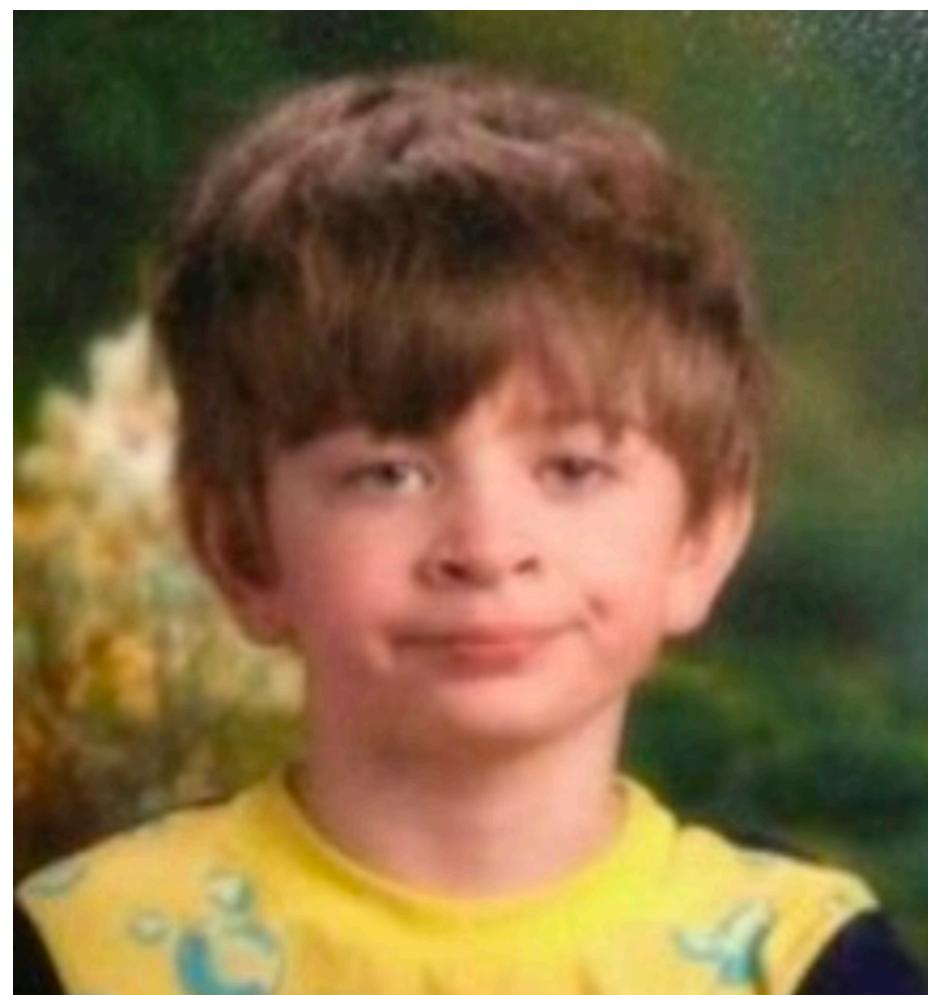
Коллеги, завтра в 12 планерка по результатам квартала

Коснитесь, чтобы ответить

Я буду

Хорошо

Это жопа



# Маруся

у нас было 2 мешка травы 75 таблеток мескалина 5 марок мощнейшей кислоты

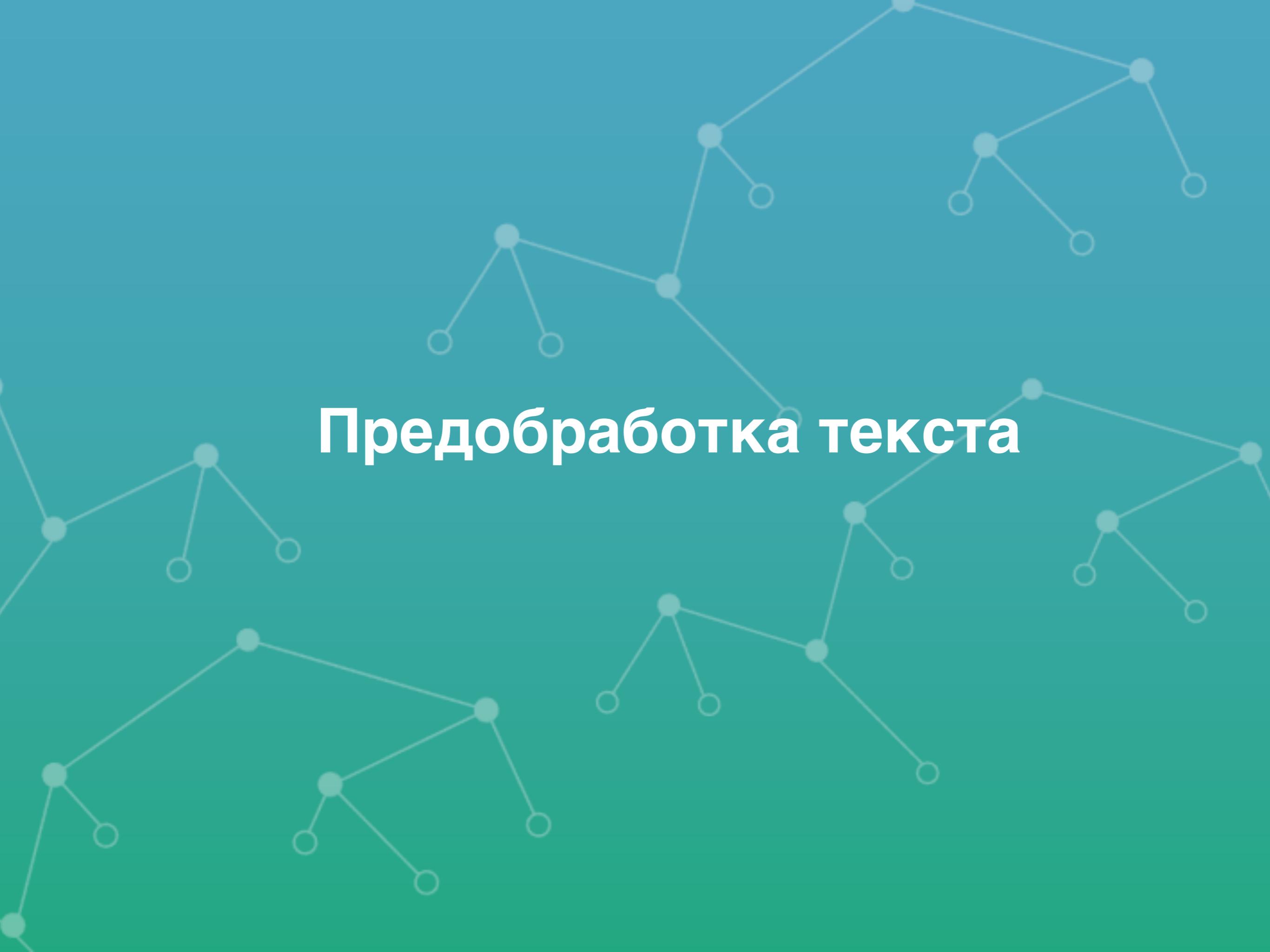
Понятно

да С какого фильма

Москва слезам не верит



# Предобработка текста



# Нормализация текста

---



**Проблема:** много словоформ и однокоренных слов

*левый, левая, левого, левых, слева...*

**Зачем нормализовать:**

- Снизить размерность признакового пространства
- Повысить качество модели

**Применяют:**

- Стемминг
- Лемматизация

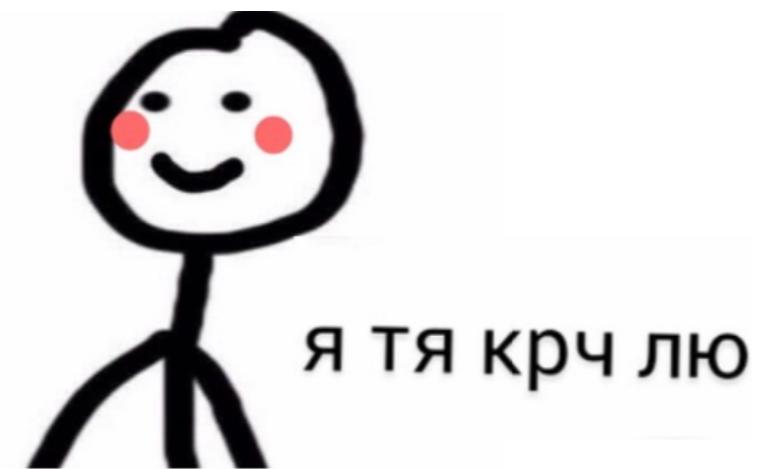
# Стемминг



Словоформа превращается в **стем** (псевдооснову), путем отбрасывания псевдоокончания.

*Бессловарный* подход: эвристики или статистика

«Мы с ним - лучши~~е~~ друзья» —> «Мы с ним - лучш друз»



# Стемминг



nltk: Porter stemmer, Snowball stemmer, ...

```
# Endings.  
  
_re_perfective_gerund = re.compile(  
    r"(((?P<ignore>[ая])(в|вши|вшились))|(ив|ивши|ившились|ыв|ывши|ывшились))$"  
)  
  
_re_adjective = re.compile(  
    r"(ее|ие|ые|ое|ими|ыми|ей|ий|ый|ой|ем|им|ым|ом|его|ого|ему|ому|их|ых|"  
    r"ую|юю|ая|яя|ою|ею)$"  
)
```

- повышает полноту
- снижает точность из-за одинаковых стемов разных слов («левый», «лев» —> «лев»)

# Лемматизация



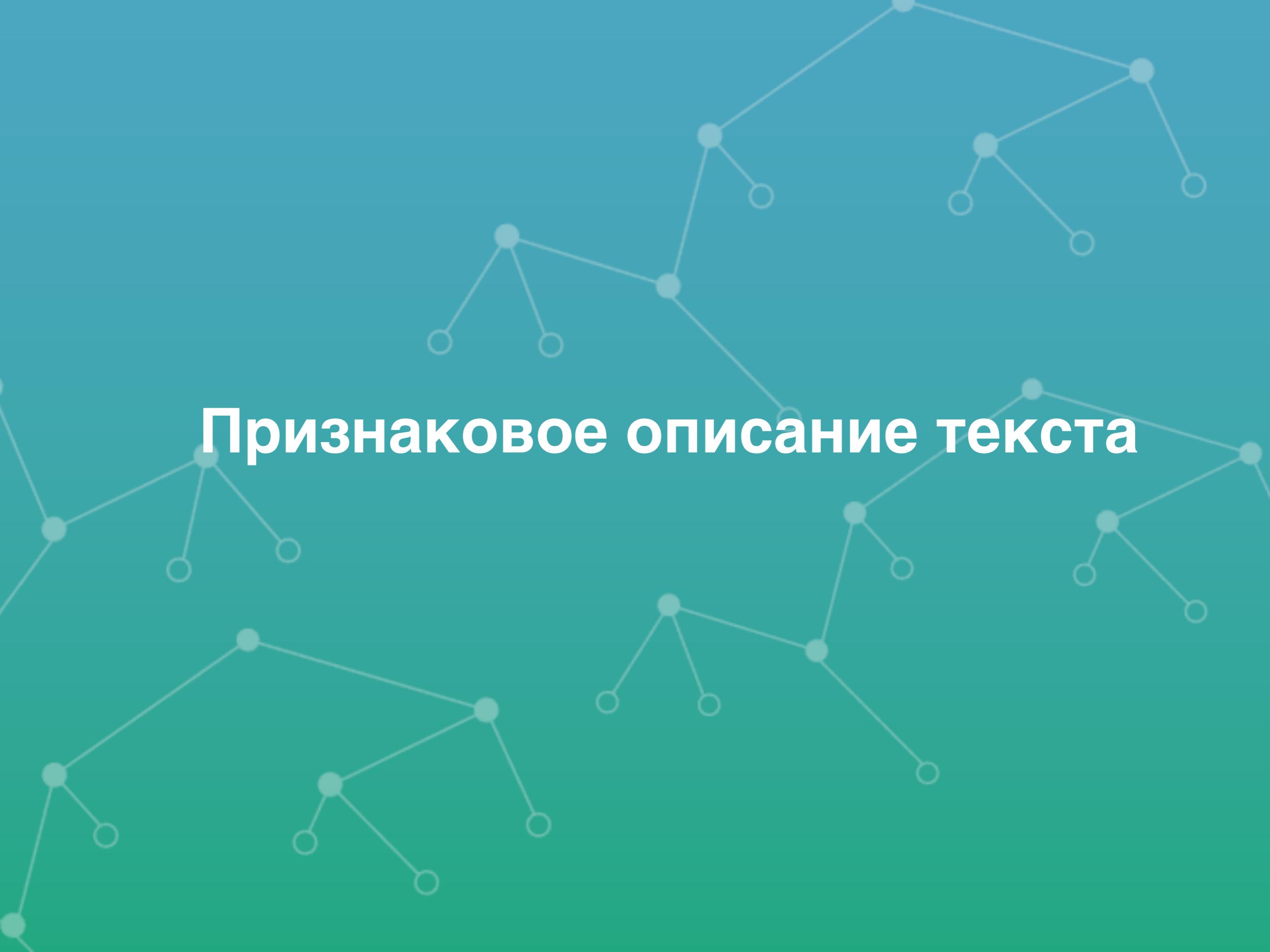
- приведение к нормальной форме слова - лемме
- словарный подход
- возможность генерации и склонения несловарных слов  
«хрюкотали» —> «хрюкотать»)



Libs:

- pymorphy2
- mystem
- АОТ

# Признаковое описание текста



# One-hot encoding



- бинарный признак наличия слова в документе
- представление документа - вектор с длиной, равной размеру словаря коллекции



# CountVectorizer

---



- хранится не бинарный признак, а счетчик (сколько раз слово встретилось в документе)
- больше информации и лучше качество
- есть проблема с частотными словами
- те же проблемы с размером вектора текста

# TfidfVectorizer

---



Слишком частотные слова отражают семантику корпуса, но не тему документа.

Слишком редкие слова - можно рассматривать как шум.

Наиболее информативные слова - встречающиеся  
довольно часто в относительно небольшом количестве  
документов.

# Term Frequency (TF)



Частота встречаемости слова в конкретном тексте корпуса

weighting scheme	tf weight
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$ <span style="color:red">←</span> Длина документа в словах
log normalization	$\log(1 + f_{t,d})$

# Inversed Document Frequency (IDF)



Document Frequency (DF) отражает, *в скольки документах корпуса* встречается данный токен.

$$DF = \frac{|\{d_i \in D | t \in d_i\}|}{|D|}$$

$$IDF = \frac{1}{DF} \text{ - inversed DF}$$

$$TfIdf = TF * IDF = TF * \log\left(\frac{|D|}{|\{d_i \in D | t \in d_i\}|}\right)$$



сглаживание

# Bag Of Words



У всех вышеперечисленных методов есть один недостаток:  
*не учитывается порядок слов в предложении*

## Документ - мешок слов (BOW)

- 
- 1) «Мне нравится, что вы больны **не** мной»
  - 2) «Мне **не** нравится, что вы больны мной»

*Один и тот же вектор для обоих предложений*

# Bag Of Ngrams



*Мешок слов (BOW) -> Мешок слов + n-грамм*

- 1) «**Мне нравится**, что вы больны **не мной**»
- 2) «**Мне не нравится**, что вы **больны мной**»

- ухватываем больше контекста
- вектор текста получается еще более жирный





## Модификация признаков: Hashing Trick

Можно перейти в новое признаковое пространство за счет хеширования признаков исходного пространства

- Выбираем количество новых признаков ( $K$  bucket'ов для хеширования)
- Выбираем хеш-функцию, отображающую признак на число  $[0..K]$
- Хешируем каждый признак исходного пространства (слова/нграммы)
- Работаем с новыми признаками



## Модификация признаков: Hashing Trick

Мы только что сократили размерность пространства признаков с  $N$  до  $K$  !



*Hashing Trick/Feature Hashing*

$$N \sim 10^5 - 10^8$$

$$K \sim 10^3 - 10^5$$

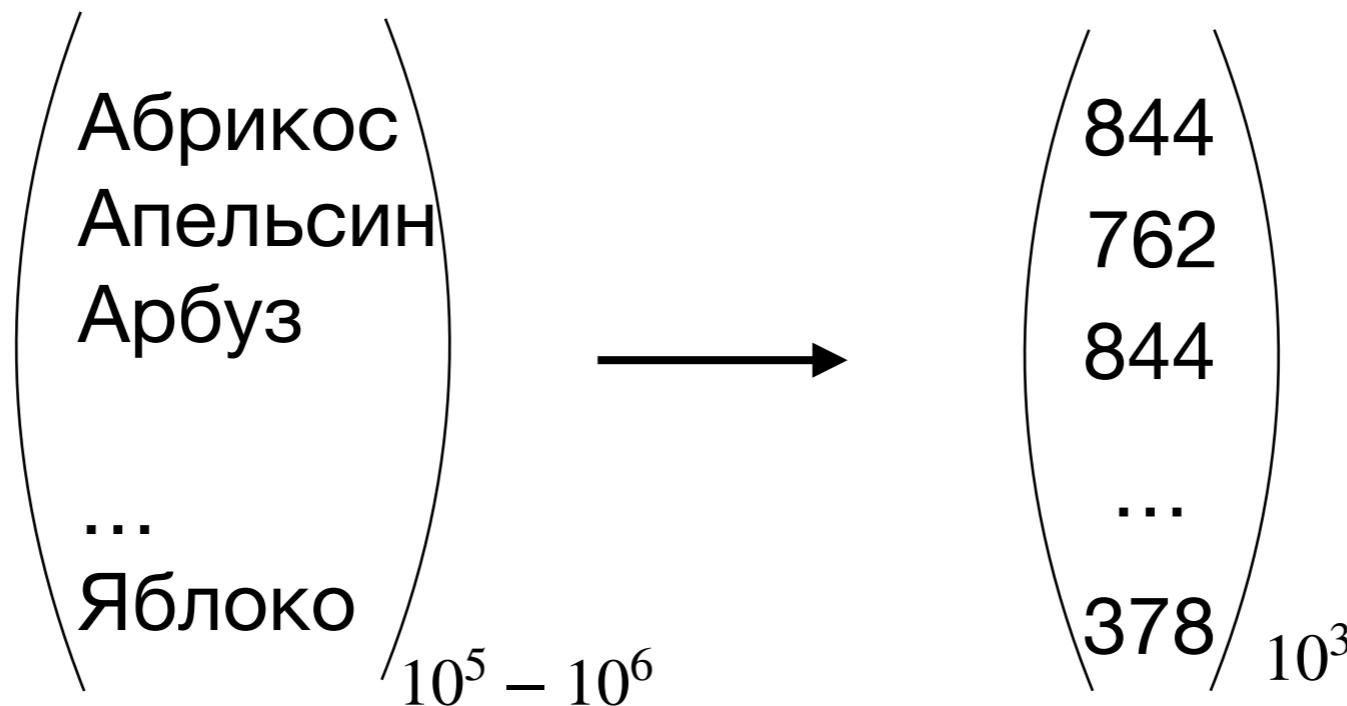
- sklearn  
(HashingVectorizer)
- VowpalWabbit
- FastText



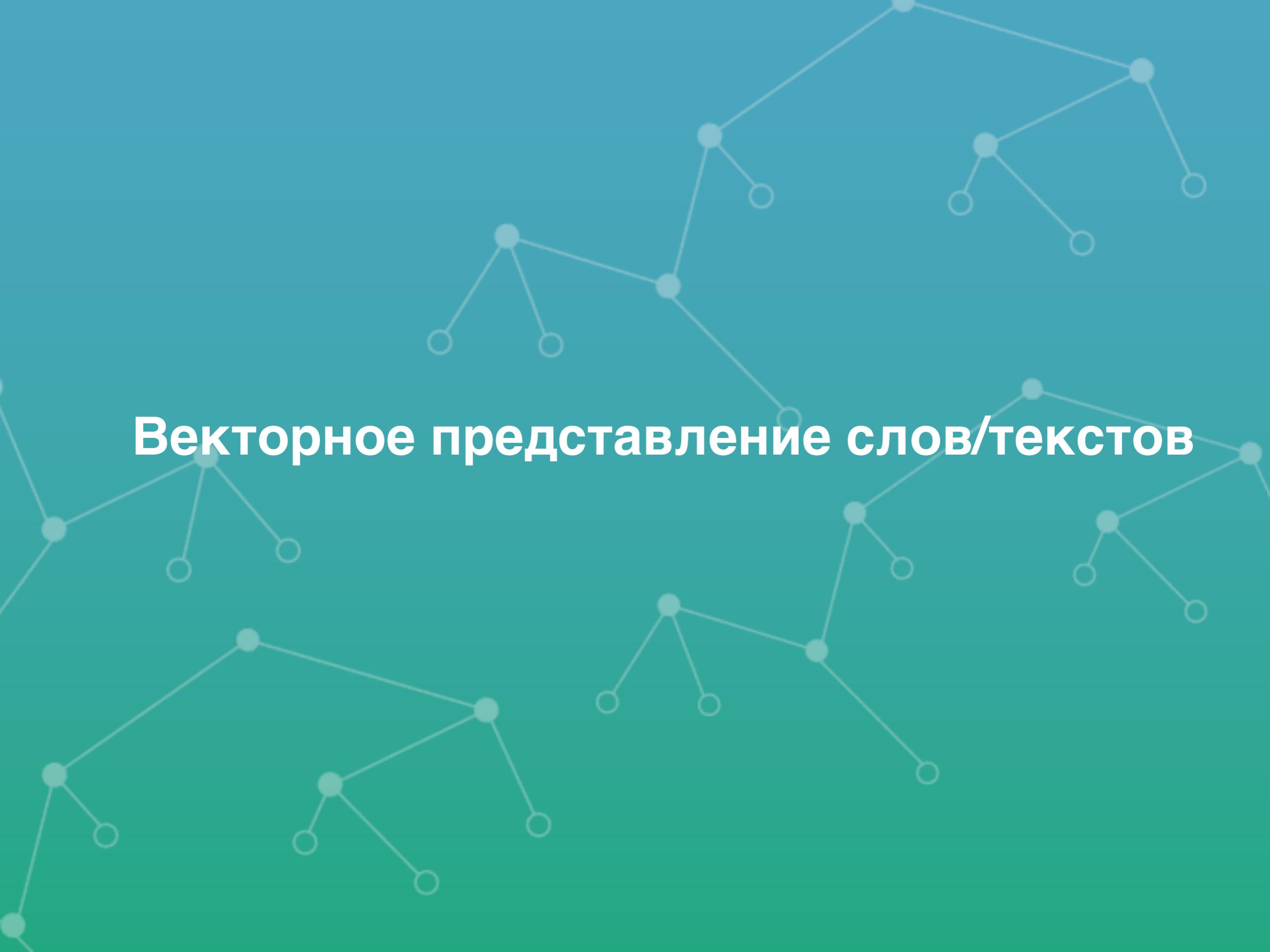
# Модификация признаков: Hashing Trick

$$K = 10^3$$

«Абрикос» → sha1(«Абрикос») mod (K) = 844



# Векторное представление слов/текстов



# Векторное представление текстов/слов

---



**Проблема:** векторное представление текста для подходов, рассмотренных выше - разреженные вектора очень большого размера

Хотим получить векторное представление для текстов/слов, чтобы:

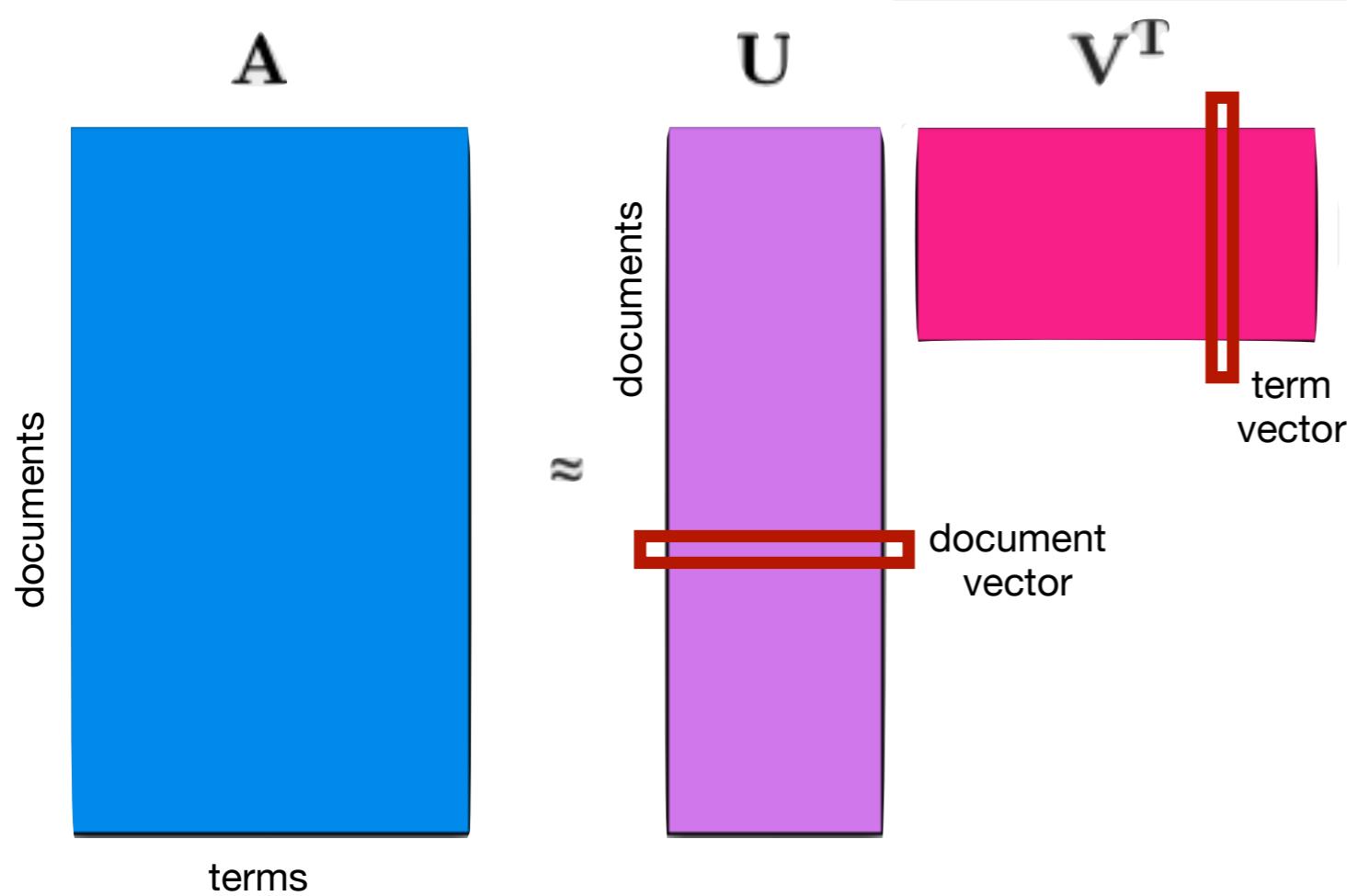
- вектора были плотные
- тексты/слова, схожие по смыслу, имели схожие вектора



# Модификация признаков: разложение матрицы

Можно перейти в новое признаковое пространство.

Например, приблизив матрицу признаков с помощью произведения матриц меньшего ранга (t-SVD, PCA и т.д.).



## Проблемы:

- вычислительно сложно
- непонятно, как работать с новыми документами

# Дистрибутивная семантика

---



Дистрибутивная гипотеза: смысл слова - *распределение над его контекстами*

- Я люблю ритмичную **X**
- Какую **X** вы предпочитаете слушать во время бега ?

Никаких затруднений восстановить пропуск - контекст определяет слово

Слова похожи по смыслу, если встречаются в похожих контекстах

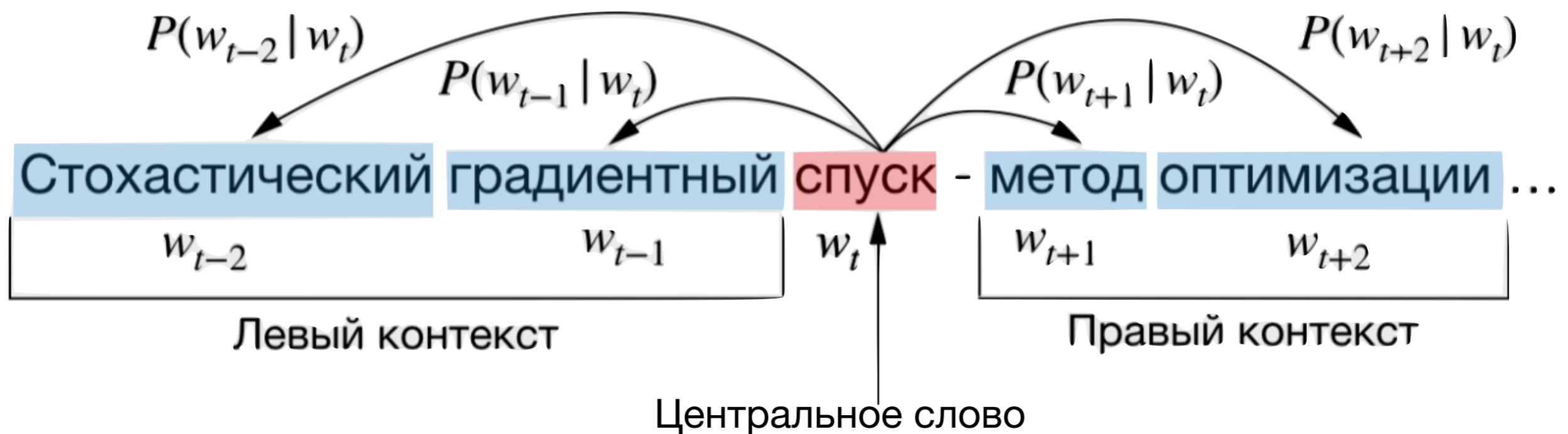
# Word2Vec на пальцах



Можно предсказывать слово по его контексту и наоборот.

Предсказываем слово по контексту - **Continuous Bag Of Words (CBOW)**

Предсказываем контекст по центральному слову - **Skip-Gram**



# Word2Vec: objective



Максимизируем правдоподобие по коллекции:

$$L(\theta) = \prod_{t=1}^T \left[ \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta) \right]$$

Контекст (BOW)

Удобнее минимизировать лосс:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

# Word2Vec: word vectors



Как оценить  $P(w_c | w_t)$ ?

Допущения:

- каждому слову соответствует векторное представление
- каждое слово выступает в двух ипостасях: как центральное и контекстное - будем использовать для каждого слова **два вектора**:

$v_w$  - когда слово  $w$  - центральное

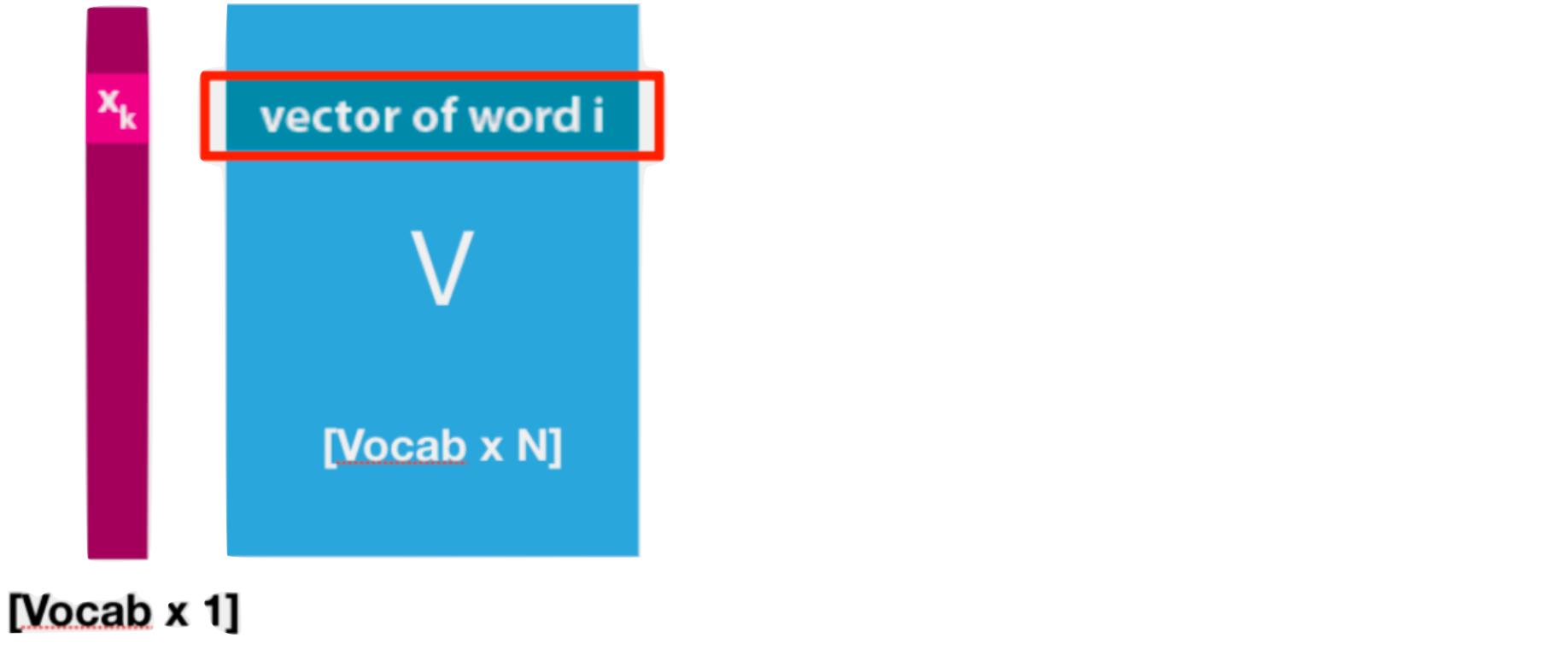
$u_w$  - когда слово  $w$  - контекстное

$u_c^T \cdot v_t$  - чем больше, тем выше вероятность встретить  $w_c$  в контексте  $w_t$

Нормализуют с помощью Softmax:

$$P(w_c | w_t) = \frac{\exp(u_c^T \cdot v_t)}{\sum_{w \in V} \exp(u_w^T \cdot v_t)}$$

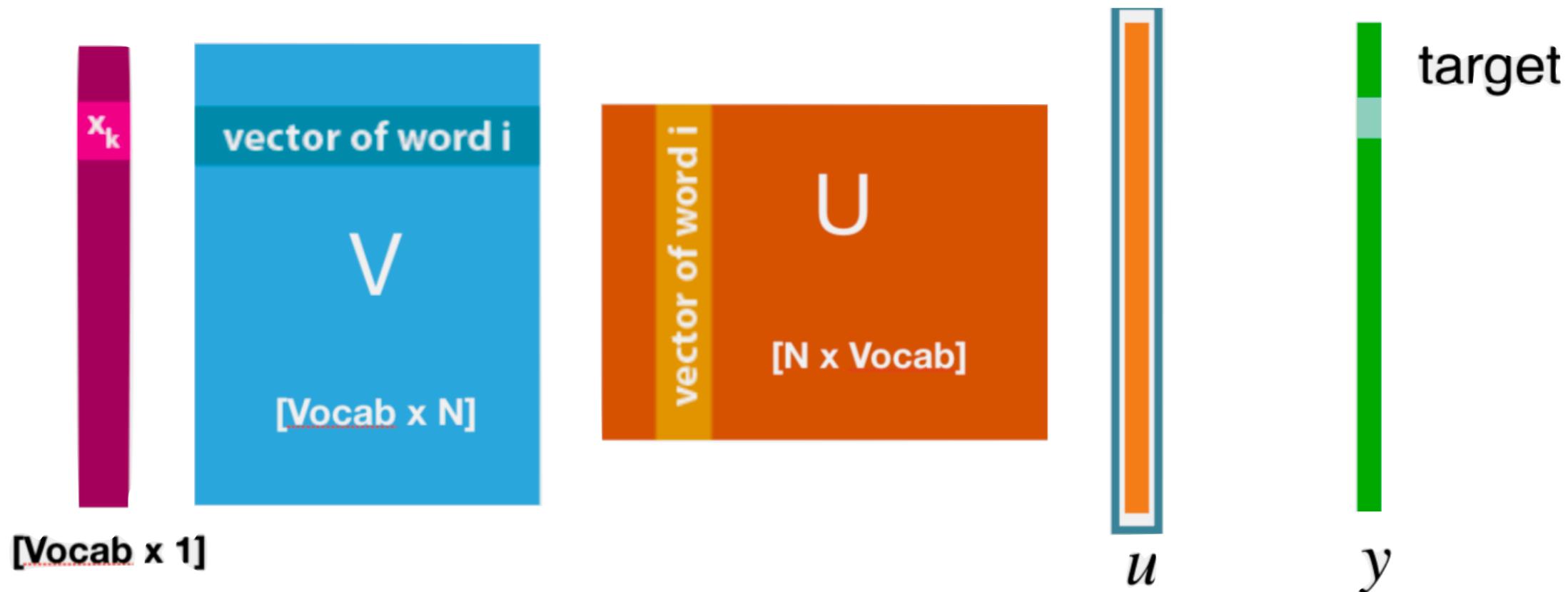
# Word2Vec: Toy Example



**Forward pass:**

$$h = V^T x = V_{(k,:)}^T := v_{w_i}^T$$

# Word2Vec: Toy Example

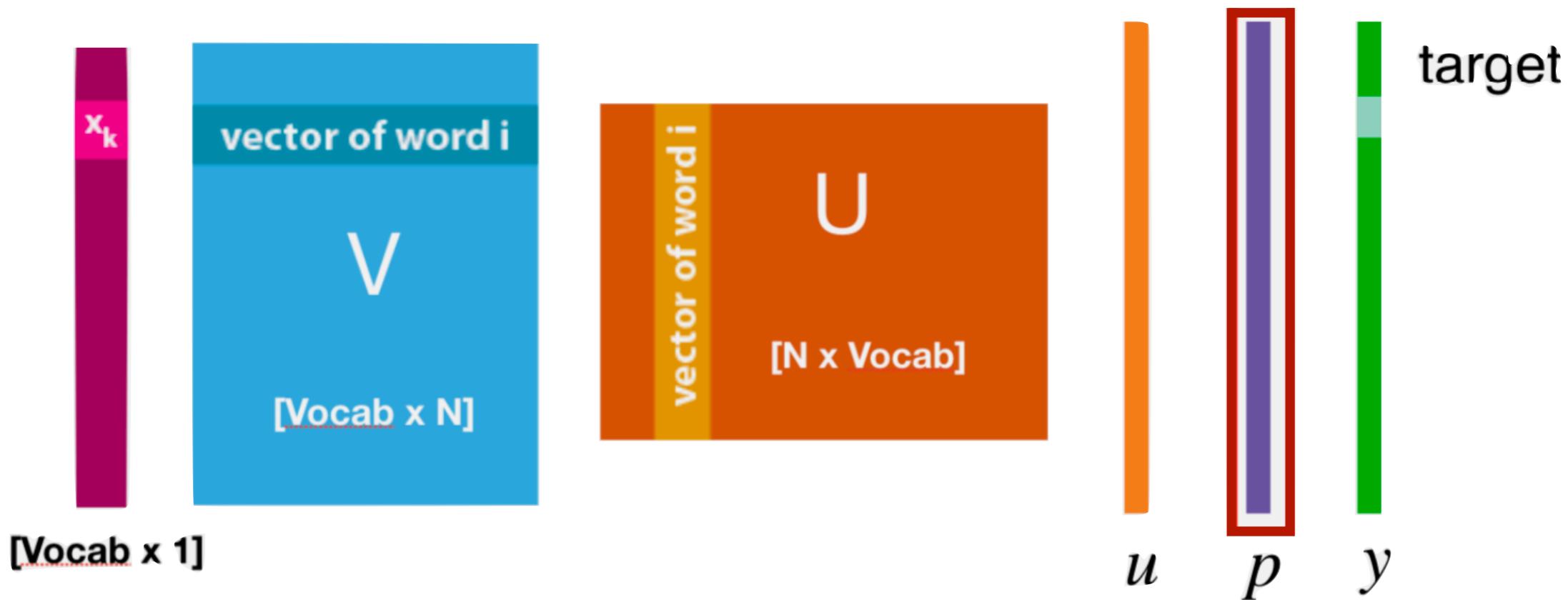


**Forward pass:**

$$h = V^T x = V_{(k,:)}^T := v_{w_i}^T$$

$$u = U^T h = U^T V^T x$$

# Word2Vec: Toy Example



**Forward pass:**

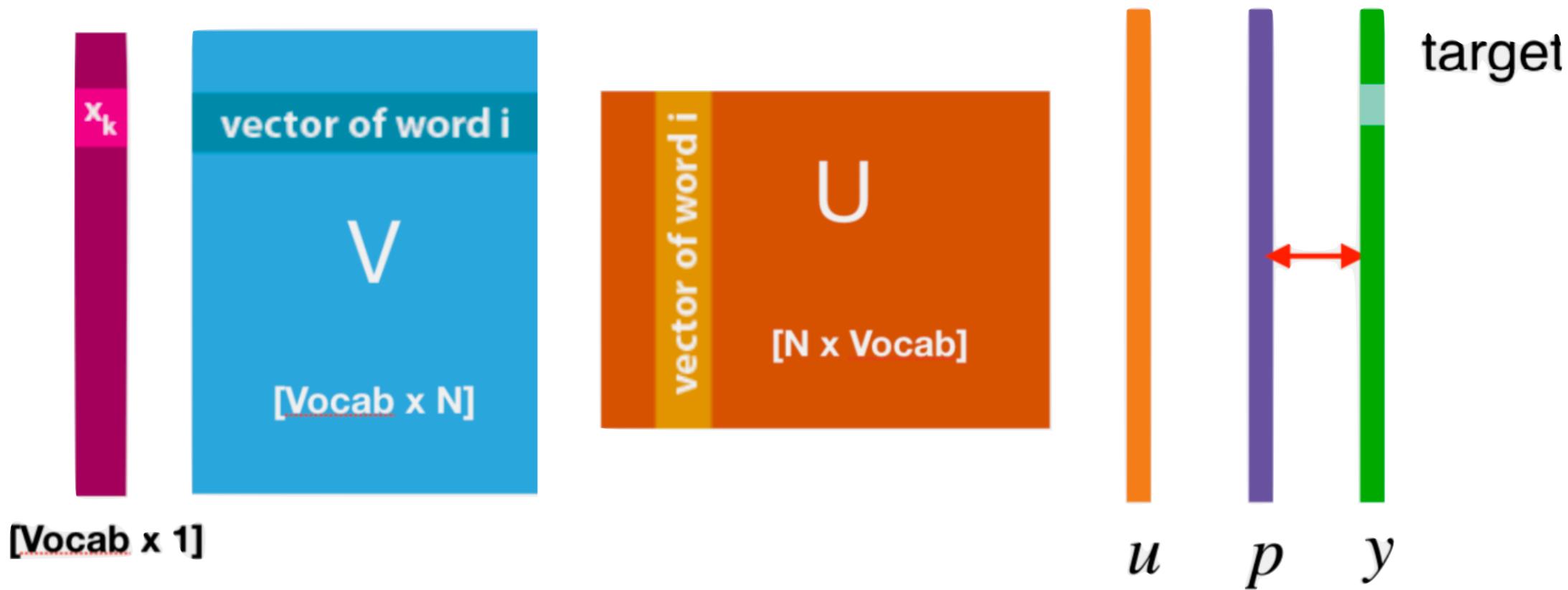
$$h = V^T x = V_{(k,:)}^T := v_{w_i}^T$$

$$u = U^T h = U^T V^T x$$

$$p = \text{Softmax}(u)$$



# Word2Vec: Toy Example



**Forward pass:**

$$h = V^T x = V_{(k,:)}^T := v_{w_i}^T$$

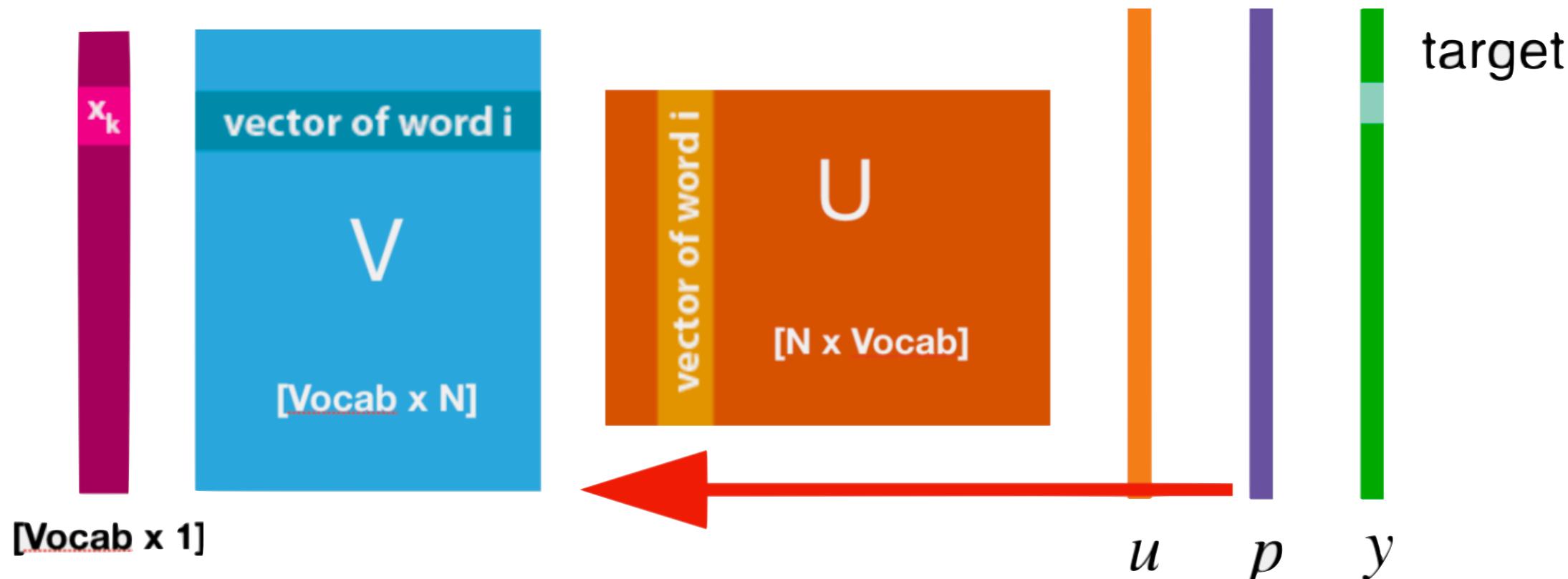
$$u = U^T h = U^T V^T x$$

$$p = \text{Softmax}(u)$$

$$\text{loss} = \text{cross\_entropy}(y, p)$$



# Word2Vec: Toy Example



**Forward pass:**

$$h = V^T x = V_{(k,:)}^T := v_{w_i}^T$$

$$u = U^T h = U^T V^T x$$

$$p = \text{Softmax}(u)$$

$$\text{loss} = \text{cross\_entropy}(y, p)$$

**Backward pass:**

Последовательно  
вычисляем градиенты по  
параметрам и изменяем их

# Skip-Gram Negative Sampling (SGNS)



$$P(w_c | w_t) = \frac{\exp(u_c^T \cdot v_t)}{\sum_{w \in V} \exp(u_w^T \cdot v_t)}$$

- Считать честный софтмакс долго - суммирование по всему словарю
- Можно использовать различные аппроксимации софтмакса, например, Hierarchical Softmax
- Или применяют SkipGram Negative Sampling (SGNS): учим модель отличать слова из контекста от рандомно взятых из словаря.

# Word2Vec: качество векторов

---



## Intrinsic evaluation («внутренняя» оценка):

- «в вакууме» на вспомогательной подзадаче
- не коррелирует с продуктными метриками
- быстро

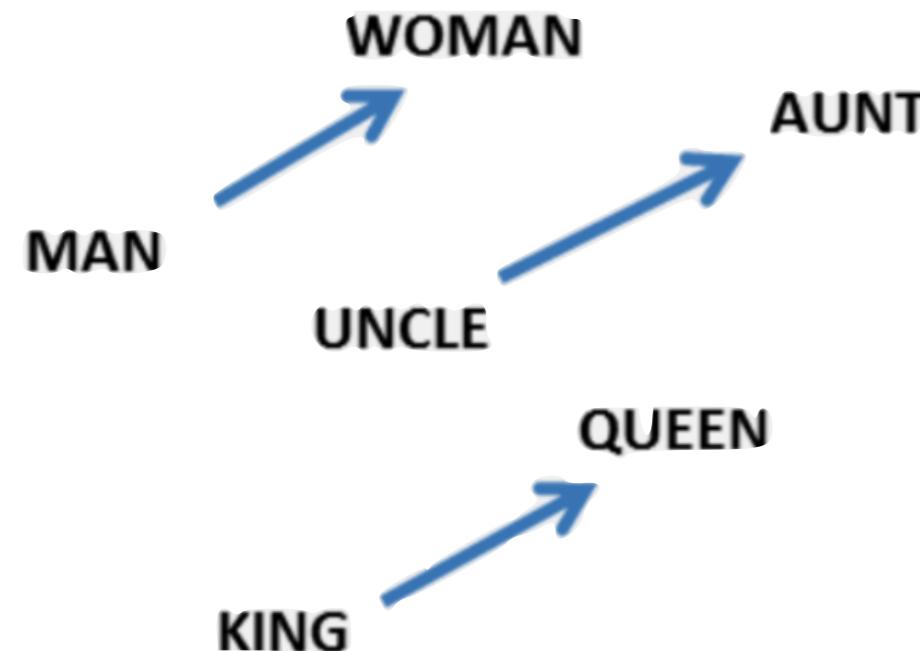
## Extrinsic evaluation («внешняя» оценка):

- на реальной задаче
- долго
- если работает плохо, то непонятно, что именно

# Word2Vec: внутренняя оценка

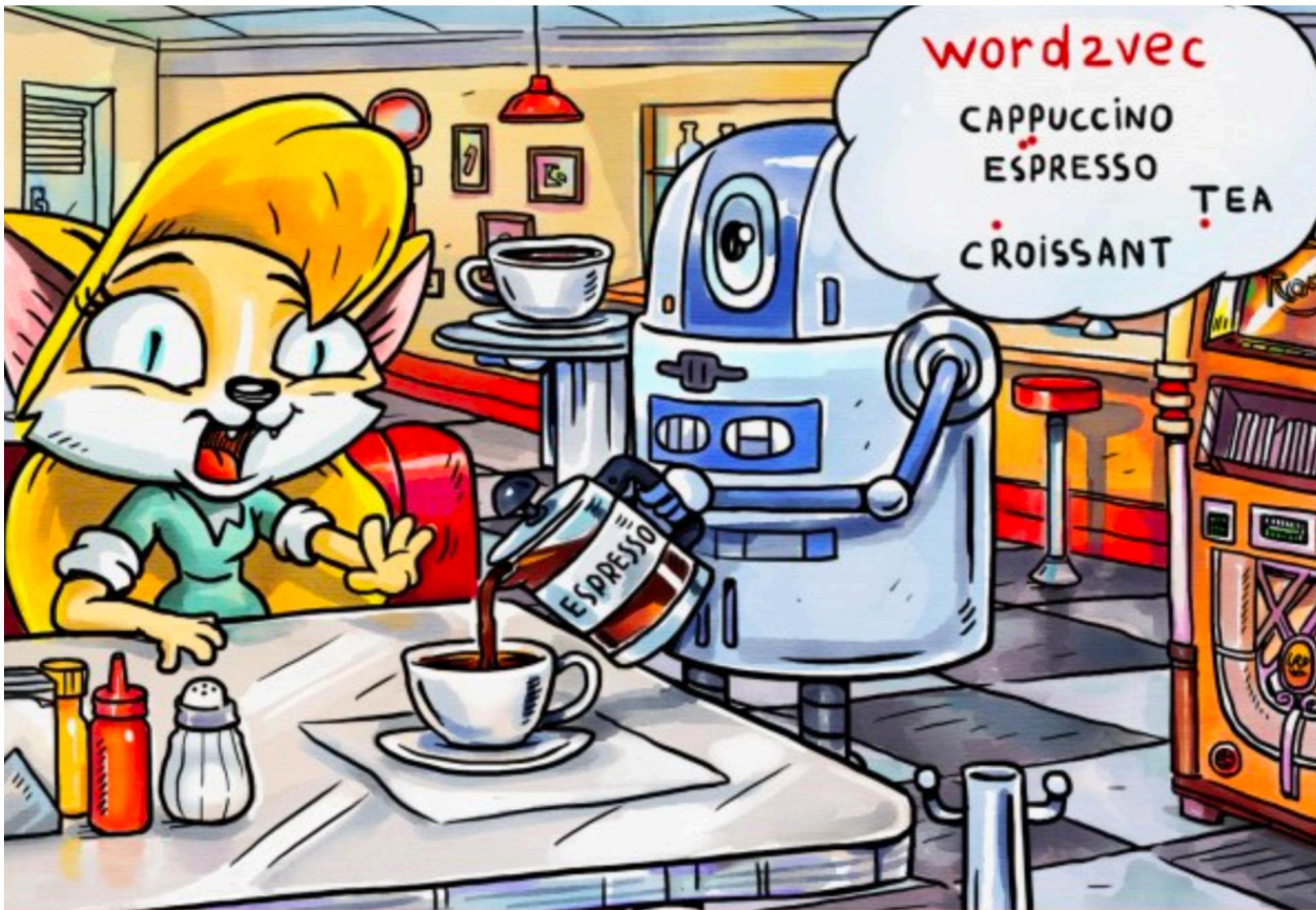


## Word Vector Analogies (semantic and syntactic)



<i>Expression</i>	<i>Nearest token</i>
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

# Word2Vec: внутренняя оценка



- Espresso? But I ordered a cappuccino!
- Don't worry, the cosine distance between them is so small that they are almost the same thing.

# Word2Vec: внешняя оценка

---



## Оценка на конкретной задаче

- Machine Translation
- NER
- Question Answering
- Information Retrieval
- etc

# FasText: Word2Vec на стероидах



## Проблемы Word2Vec: out-of-vocabulary (OOV) tokens



- Представляем слово в виде мешка символьных n-грамм разного порядка
- Учим эмбеддинги для каждой n-граммы вместе со словами
- Суммируем эмбеддинги для всех n-грамм слова и выученного вектора слова и получаем результирующий эмбеддинг слова

# FastText: особенности реализации

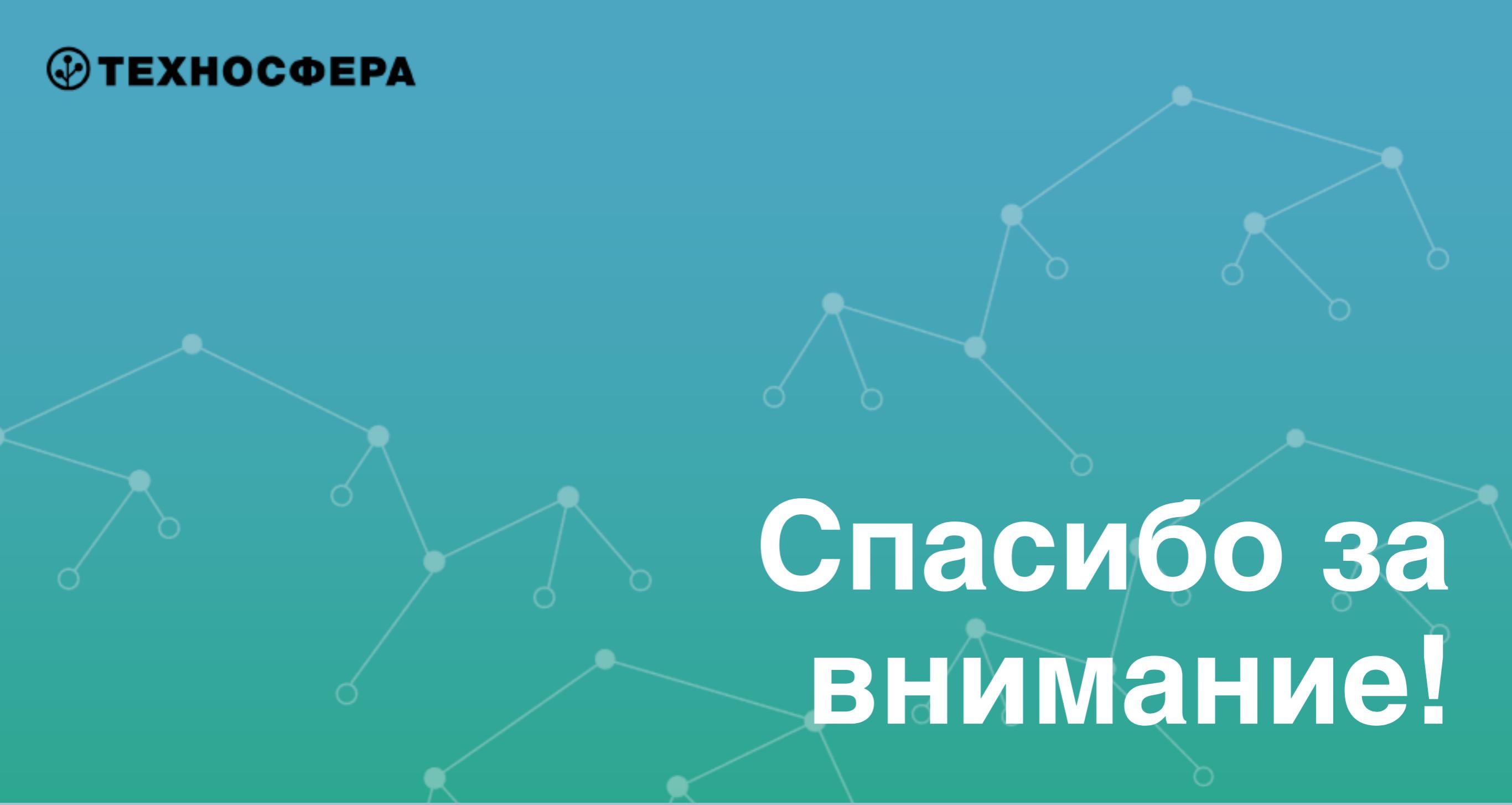


- Хеширует символные нграммы
- И словесные нграммы тоже хеширует (причем, в те же бакеты)
- Слова не хешируются
- soft-sliding window - рандомно сэмплируется размер окна из равномерного распределения
- Учится с SGNS аналогично Word2Vec
- Можно учить вектора supervised
- Можно получить сжатую версию модели (весит меньше на порядок с минимальной деградацией качества)

## FastText: плюсы

- работает со словами вне словаря
- качественные вектора для редких слов
- multi-core
- really fast!





Спасибо за  
внимание!