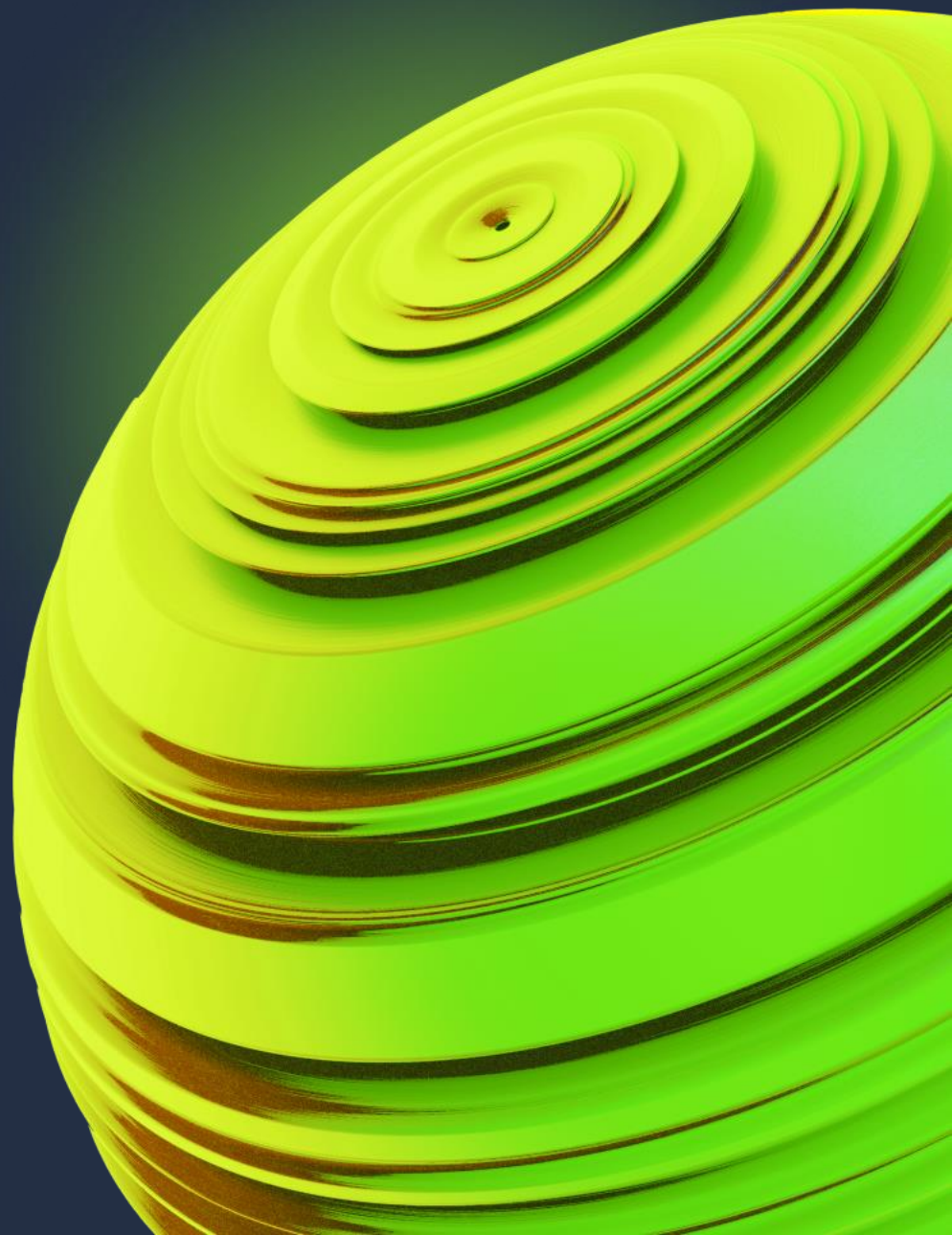
ИНСТИТУТ ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
УНИВЕРСИТЕТА ИННОПОЛИС

УНИВЕРСИТЕТ ИННОПОЛИС

Упрощенное представление текста. TF-IDF, Word2Vec

Корнеева Елена
e.korneeva@innopolis.ru, https://t.me/Allyonzy

2024



План занятия (лекция + семинар)

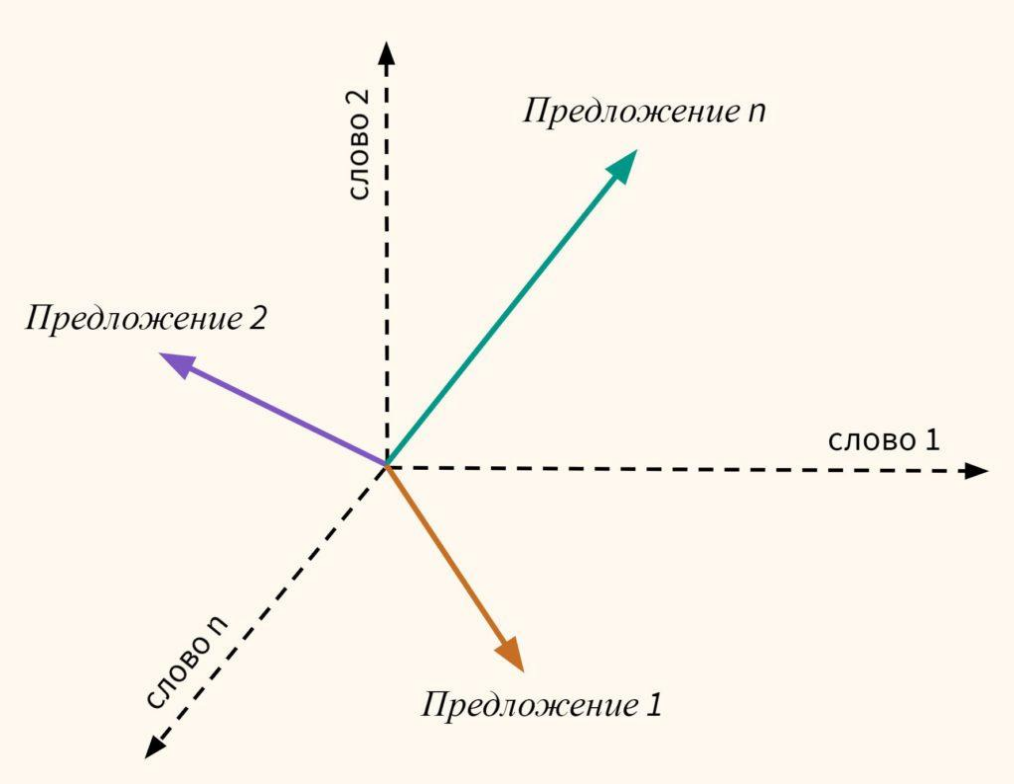
1. Векторное представление текста
2. Модель «Мешок слов» (BOW)
3. Модель TF*IDF
4. Word2Vec CBOW и Skip-gram

Векторное представление слов

Векторное представление слов (англ. word embedding) — общее название для различных подходов к моделированию языка, направленных на *сопоставление словам из некоторого словаря векторов небольшой размерности*.

Как представлять текст в компьютере?

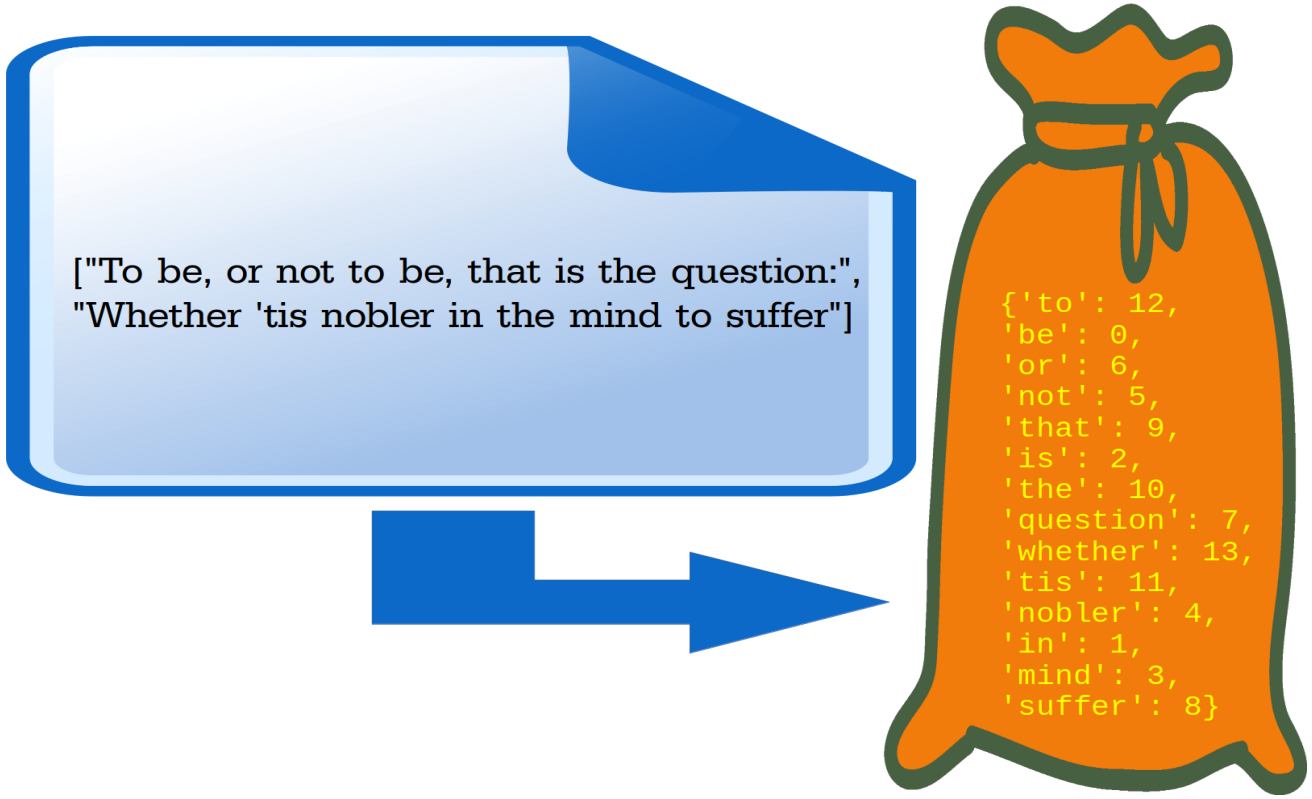
Текст – набор слов. Как представлять слова?



Мешок слов (Bag of Words, BoW)

Мешок слов (англ. bag-of-words) — это упрощенное представление текста

Текст (одно предложение или весь документ) представляется в виде мешка (мультимножества) его слов *без какого-либо учета семантики, синтаксиса и порядка слов*, но с *сохранением информации об их количестве*



Мешок слов (Bag of Words, BoW)

при правильной предобработке текста (в первую очередь удалении стоп-слов, которые и будут наиболее частотными) этот метод показывает неплохие результаты.

BOW с помощью класса Counter модуля Collection

```
# из модуля collections импортируем класс Counter
from collections import Counter

# применяем класс Counter к словам после лемматизации
# на выходе нам возвращается словарь { слово : его частота в тексте }
bow_counter = Counter(lemmatized)
# print(bow_counter)

# функция most_common() упорядочивает словарь по значению
# посмотрим на первые 10 наиболее частотных слов
print(bow_counter.most_common(10))
```

Мешок слов (Bag of Words, BoW)

Этот же метод можно реализовать с помощью класса CountVectorizer библиотеки Scikit-learn

```
# импортируем класс CountVectorizer из библиотеки Scikit-learn
from sklearn.feature_extraction.text import CountVectorizer

# создаём объект этого класса и
# указываем, что хотим перевести слова в нижний регистр, а также
# отфильтровать стоп-слова через stop_words = {'english'}
vectorizer = CountVectorizer(analyzer = "word",
                             lowercase = True,
                             tokenizer = None,
                             preprocessor = None,
                             stop_words = {'english'},
                             max_features = 5000)
```

```
# применяем этот объект к предложениям (ещё говорят документам)
bow_cv = vectorizer.fit_transform(sentences)

# на выходе получается матрица csr
print(type(bow_cv))
```

```
# для этого можно использовать .toarray()
print(bow_cv.toarray())
```

One-Hot Encoding

Признаковое пространство в one-hot векторах имеет размерность, равную мощности словаря коллекции текстов. Для каждого слова в вектор-столбце “зарезервировано” единицей свое место.

***Проще говоря**: берем вектор, длина которого равна длине словаря, и на все места ставим нули, кроме того места, номер которого совпадает с номером слова в словаре.*

Island	Biscoe	Dream	Torgensen
Biscoe	1	0	0
Torgensen	0	0	1
Dream	0	1	0

5 → one-hot encoding

категории	0	1	2	3	4	5	6	7	8	9
кодировка	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**

Подходы к векторизации текста

- **TF-IDF.** *Это статистическая мера, используемая для оценки важности слова в контексте документа, который является частью коллекции документов или корпуса. Представляет каждый документ как вектор, где координата соответствует частоте уникального слова из всего корпуса в этом документе, нормированной на обратную частоту этого слова во всем корпусе. Слова, которые редко встречаются в корпусе, но часто встречаются в одном документе, будут иметь большую важность.*
- **Word2Vec.** *Это нейронная сетевая модель для представления слов в векторном пространстве. Учитывает контекст слова в предложении, что позволяет ему улавливать семантические и синтаксические отношения между словами. Обучает векторы так, чтобы слова, встречающиеся в похожих контекстах, были ближе друг к другу в векторном пространстве. Это приводит к интересным свойствам: например, векторное отношение между "король" и "мужчина" может быть подобно векторному отношению между "королева" и "женщина".*

Подходы к векторизации текста

- **TF-IDF**

✓ *прост в понимании и имплементации*

✓ *хорошо работает для задач классификации документов и поиска документов*

- **Word2Vec**

✓ *улавливает семантические и синтаксические отношения между словами*

✓ *обучение требует значительно больше времени и вычислительных ресурсов*

TF-IDF. TF (Term Frequency)

1. “Кот съел рыбу”
2. “Кот любит рыбу”
3. “Собака любит кота”

$$tf(this, d_1) = \frac{\text{частотность слова}}{\text{всего слов}}$$

$$tf(this, d_2) = \frac{\text{частотность слова}}{\text{всего слов}}$$

	кот	съел	рыбу	любит	собака
1	1	1	1	0	0
2	1	0	1	1	0
3	1	0	0	1	1

TF-IDF. IDF (Inverse Document Frequency)

	кот	съел	рыбу	любит	собака
1	1	1	1	0	0
2	1	0	1	1	0
3	1	0	0	1	1

$$idf(this, D) = log\left(\frac{всего\ документов}{документов\ с\ токеном}\right)$$

	IDF
кот	log(3 / 3) = 0
съел	log(3 / 1) = 1.098
рыбу	log(3 / 2) = 0.405
любит	log(3 / 2) = 0.405
собака	log(3 / 1) = 1.098

TF-IDF

1. “Кот съел рыбу”
2. “Кот любит рыбу”
3. “Собака любит кота”

$$TF \cdot IDF$$

$$tf - idf(this, d_1, D) =$$

$$tf - idf(this, d_2, D) =$$

	кот	съел	рыбу	любит	собака
1	0	1.098	0.405	0	0
2	0	0	0.405	0.405	0
3	0	0	0	0.405	1.098

TF-IDF

В итоге каждый документ представлен вектором TF-IDF значений его слов

Эти векторы можно использовать для сравнения документов между собой или для обучения моделей машинного обучения

	кот	съел	рыбу	любит	собака
1	0	1.098	0.405	0	0
2	0	0	0.405	0.405	0
3	0	0	0	0.405	1.098

TF-IDF с помощью библиотеки Scikit-learn

Способ 1.
CountVectorizer +
TfidfTransformer

1) TF или частоту слов мы можем взять из расчета (BOW)

2) Теперь нужно рассчитать IDF

```
# импортируем TfidfTransformer (CountVectorizer уже импортирован)
from sklearn.feature_extraction.text import TfidfTransformer

# создадим объект класса TfidfTransformer
tfidf_trans = TfidfTransformer(smooth_idf = True, use_idf = True)

# и рассчитаем IDF слов
tfidf_trans.fit(bow_cv)

# поместим результат в датафрейм
df_idf = pd.DataFrame(tfidf_trans.idf_, index = tokens, columns = ["idf_weights"])
```

3) Остается TF x IDF

```
# рассчитаем TF-IDF (по сути умножим TF на IDF)
tf_idf_vector = tfidf_trans.transform(bow_cv)
tf_idf_vector
```

TF-IDF с помощью библиотеки Scikit-learn

Способ 2. Tfidfvectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Задаем коллекцию документов
documents = [
    'Кот съел рыбу',
    'Кот любит рыбу',
    'Собака любит кота'
]

# Инициализируем TfidfVectorizer
vectorizer = TfidfVectorizer()

# Вычисляем TF-IDF для каждого документа в коллекции
tfidf_matrix = vectorizer.fit_transform(documents)

# Результат - это матрица TF-IDF, где строки соответствуют документам, а
столбцы - словам
print(tfidf_matrix.toarray())

print(vectorizer.get_feature_names_out())
```

Word2Vec

Двухслойная нейронная сеть, которая обучается представлять слова в векторном пространстве. Обучение происходит на "центральном слове" и "контекстном слове"

CBOW (Continuous Bag of Words)

Кроме двух слов перед целевым, можно учитывать ещё два слова после него. Рассматривает текст как мешок слов с контекстом

Skip-gram

Работает как скользящее окно (фактически создает n отдельных образцов в наборе данных обучения), пытается угадать соседние слова по текущему слову

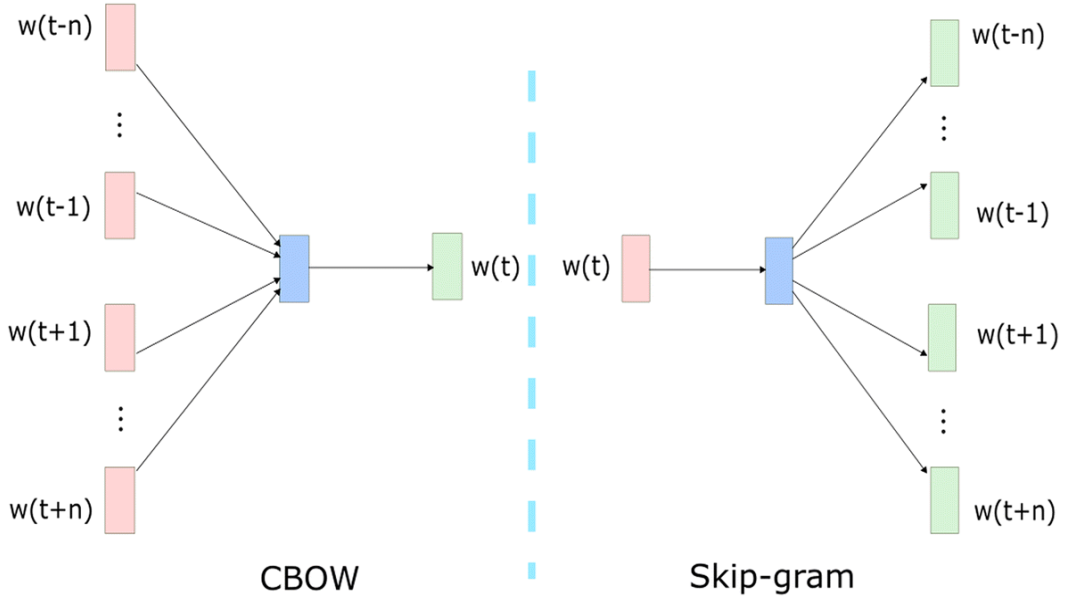
Word2Vec

CBOW
(Continuous Bag of Words)

обучается по паре (контекст, центральное слово) и пытается предсказать центральное слово, основываясь на его контексте.

Skip-gram

обучается по паре (центральное слово, контекст) и пытается предсказать контекст, основываясь на центральном слове.



Word2Vec

CBOW
(Continuous Bag of Words)

Jay was hit by a _____ bus in...

input 1	input 2	input 3	input 4	output
by	a	bus	in	red

Skip-gram

Jay was hit by a red bus in...

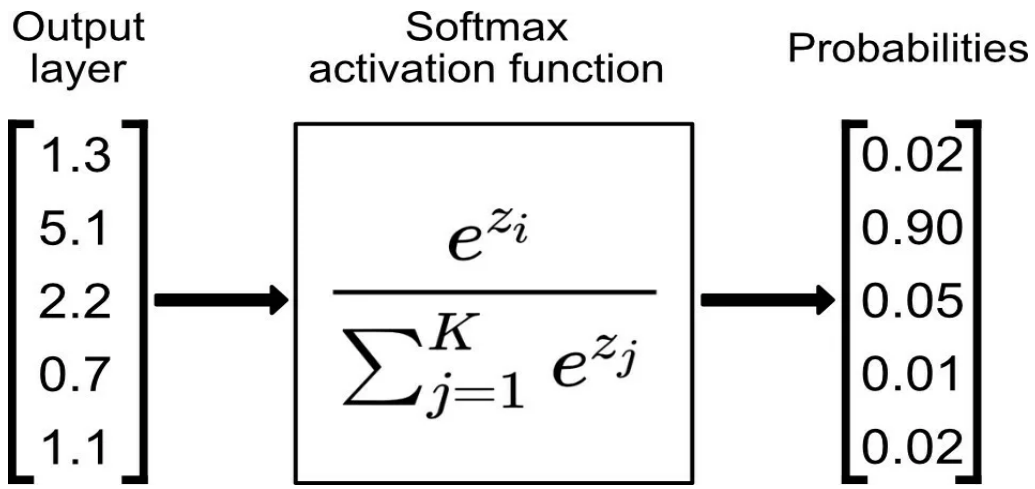
input	output
red	by
red	a
red	bus
red	in

Negative Sampling

Многие слова в текстах не встречаются вместе, поэтому модель выполняет много лишних вычислений

Подсчёт softmax — вычислительно дорогая операция

Negative Sampling позволяет *максимизировать* ***вероятность*** *встречи нужного слова в контексте, который является для него типичным, и минимизировать – в редком/нетипичном контексте.*



Negative Sampling

Vanilla Skip-Gram

W_output (old)				Learning R.		grad_W_output				W_output (new)		
-0.560	0.340	0.160		0.05	×	0.064	0.071	-0.014	=	**-0.563**	**0.336**	**0.161**
-0.910	-0.440	1.560				0.098	0.015	0.063		**-0.915**	**-0.441**	**1.557**
-1.210	-0.130	-1.320				0.069	0.089	0.045		**-1.213**	**-0.134**	**-1.322**
1.670	-0.150	-1.030				0.014	0.085	0.079		**1.669**	**-0.154**	**-1.034**
1.720	-1.460	0.730				-0.021	0.067	0.071		**1.721**	**-1.463**	**0.726**
0.000	1.390	-0.120				-0.098	-0.088	0.091		**0.005**	**1.394**	**-0.125**
-0.060	1.520	-0.790				-0.072	-0.078	-0.089		**-0.056**	**1.524**	**-0.786**
0.800	1.850	-1.670				0.046	-0.079	-0.053		**0.798**	**1.854**	**-1.667**
-1.370	1.320	-0.480				-0.049	-0.087	0.025		**-1.368**	**1.324**	**-0.481**
0.670	1.990	-1.850				-0.060	0.092	0.042		**0.673**	**1.985**	**-1.852**
-1.520	-1.740	-1.860				0.074	0.050	0.070		**-1.524**	**-1.743**	**-1.864**
(11X3)						(11X3)				(11X3)		

Negative Sampling

W_output (old)				Learning R.		grad_W_output				W_output (new)		
-0.560	0.340	0.160		0.05	×	Not computed!			=	-0.560	0.340	0.160
-0.910	-0.440	1.560								-0.910	-0.440	1.560
-1.210	-0.130	-1.320								-1.210	-0.130	-1.320
1.670	-0.150	-1.030								1.670	-0.150	-1.030
1.720	-1.460	0.730								1.720	-1.460	0.730
0.000	1.390	-0.120								0.000	1.390	-0.120
-0.060	1.520	-0.790								-0.060	1.520	-0.790
0.800	1.850	-1.670		Positive sample, w_o		0.031	0.030	0.041		**0.798**	**1.849**	**-1.672**
-1.370	1.320	-0.480		Negative sample, k=1		-0.090	0.031	-0.065		**-1.366**	**1.318**	**-0.477**
0.670	1.990	-1.850		Negative sample, k=2		0.056	0.098	-0.061		**0.667**	**1.985**	**-1.847**
-1.520	-1.740	-1.860		Negative sample, k=3		0.069	0.084	-0.044		**-1.523**	**-1.744**	**-1.858**
(11X3)						(11X3)				(11X3)		

Реализация Word2Vec с Gensim

```python
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess

# Исходные текстовые документы
documents = [
    'Кот съел рыбу',
    'Кот любит рыбу',
    'Собака любит кота'
]

# Предобработка и токенизация документов
tokenized_docs = [simple_preprocess(doc) for doc in documents]

# Создание и обучение модели Word2Vec
model = Word2Vec(sentences=tokenized_docs, vector_size=50, window=5,
min_count=1, workers=4)

# Сохранение модели
model.save("word2vec.model")
```

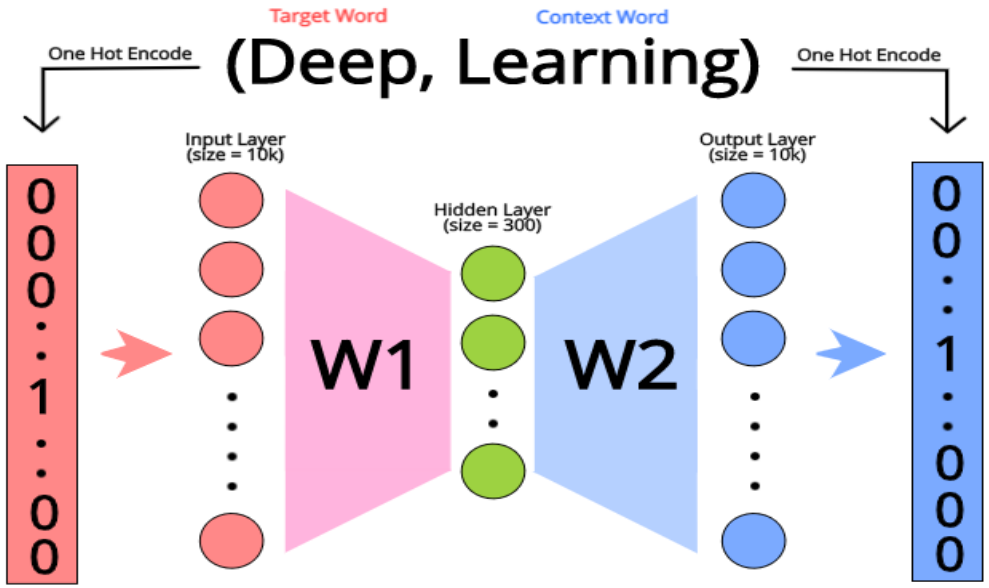
Реализация Word2Vec с Gensim

```python
# Для получения вектора конкретного слова
vector = model.wv['кот']
print(vector)

# Для получения наиболее близких слов к данному
similar_words = model.wv.most_similar('кот', topn=5)
print(similar_words)
```

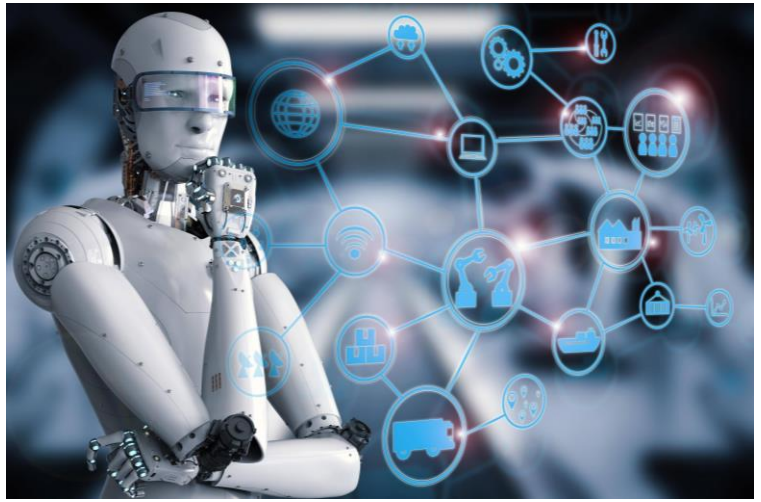
ДЕМОНСТРАЦИЯ

Пример работы с библиотеками
для работы с текстом как
векторным представлением



Полезные ссылки

1. Обработка естественного языка. URL: https://www.dmitrymakarov.ru/intro/topic-identification-19/
2. Word2vec в картинках. URL: https://habr.com/ru/articles/446530/
3. Реализация Word2Vec с библиотекой Gensim на Python URL: https://tonais.ru/library/realizatsiya-word2vec-s-bibliotekoy-gensim-na-python
4. Google News и Лев Толстой: визуализация векторных представлений слов с помощью t-SNE. URL: https://habr.com/ru/companies/vk/articles/426113/
5. Репозиторий курса по анализу текстов и обработке естественного языка на ФИВТ МФТИ. URL: https://github.com/andybelov/nlp_mipt
6. Векторное представление слов. URL: https://neerc.ifmo.ru/wiki/index.php?title=Векторное_представление_слов
7. Нейросети для работы с последовательностями. URL: https://academy.yandex.ru/handbook/ml/article/nejroseti-dlya-raboty-s-posledovatelnostyami
8. Обзор четырёх популярных NLP-моделей. URL: https://proglib.io/p/obzor-chetyreh-populyarnyh-nlp-modeley-2020-04-21



ИНСТИТУТ ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
УНИВЕРСИТЕТА ИННОПОЛИС

Спасибо за внимание!

Telegram

E-mail

Контакты

Корнеева Елена

https://t.me/Allyonzy

e.korneeva@innopolis.ru