



Deep Learning School

Физтех-Школа Прикладной математики и информатики (ФПМИ) МФТИ

Some parts of the notebook are almost the copy of [mmta-team course](#). Special thanks to mmta-team for making them publicly available. [Original notebook](#).

Прочитайте семинар, пожалуйста, для успешного выполнения домашнего задания. В конце ноутка напишите свой вывод. Работа без вывода оценивается ниже.

Задача поиска схожих по смыслу предложений

Мы будем ранжировать вопросы [StackOverflow](#) на основе семантического векторного представления

До этого в курсе не было речи про задачу ранжирования, поэтому введем математическую формулировку

Задача ранжирования(Learning to Rank)

- X - множество объектов
 - $X^I = \{x_1, x_2, \dots, x_I\}$ - обучающая выборка
- На обучающей выборке задан порядок между некоторыми элементами, то есть

нам известно, что некий объект выборки более релевантный для нас, чем другой:

- $i \prec j$ - порядок пары индексов объектов на выборке X^I с индексами i и j

Задача:

построить ранжирующую функцию $a : X \rightarrow R$ такую, что $i \prec j \Rightarrow a(x_i) < a(x_j)$

Ranking



Embeddings

Будем использовать предобученные векторные представления слов на постах Stack Overflow.

[A word2vec model trained on Stack Overflow posts](#)

```
In [64]: import requests, shutil
from pathlib import Path

url = "https://zenodo.org/record/1199620/files/SO_vectors_200.bin?download=1"
out_path = "SO_vectors_200.bin"

if not Path(out_path).exists():
    with requests.get(url, stream=True) as r:
        r.raise_for_status()
        with open(out_path, "wb") as f:
            shutil.copyfileobj(r.raw, f)

print("Saved to", out_path)
```

Saved to SO_vectors_200.bin
 ERROR! Session/line number was not unique in database. History logging moved to new session 26

```
In [65]: from gensim.models.keyedvectors import KeyedVectors
wv_embeddings = KeyedVectors.load_word2vec_format("SO_vectors_200.bin", binary=True)
```

Как пользоваться этими векторами?

Посмотрим на примере одного слова, что из себя представляет embedding

```
In [66]: word = 'dog'
if word in wv_embeddings:
    print(wv_embeddings[word].dtype, wv_embeddings[word].shape)

float32 (200,)
```

```
In [67]: print(f"Num of words: {len(wv_embeddings.index_to_key)})")

Num of words: 1787145
```

Найдем наиболее близкие слова к слову `dog`:

Вопрос 1:

- Входит ли слово `cat` в топ-5 близких слов к слову `dog`? Какое место оно занимает?

```
In [68]: topn = 5
target = 'cat'

similar = wv_embeddings.most_similar(word, topn=topn)
ranks = [w for w, _ in similar]
if target in ranks:
    print(f"Да, '{target}' в топ-{topn}. Место: {ranks.index(target) + 1}")
else:
    print(f"'{target}' не входит в топ-{topn} ближайших слов к '{target}'.")
```

'cat' не входит в топ-5 ближайших слов к 'cat'.

Ваш ответ: 'cat' не входит в топ-5 ближайших слов к 'dog'.

Векторные представления текста

Перейдем от векторных представлений отдельных слов к векторным представлениям вопросов, как к **среднему** векторов всех слов в вопросе. Если для какого-то слова нет предобученного вектора, то его нужно пропустить. Если вопрос не содержит ни одного известного слова, то нужно вернуть нулевой вектор.

```
In [69]: from abc import ABC, abstractmethod

class Tokenizer(ABC):
    @abstractmethod
    def tokenize(self, text:str)->list[str]: ...
```

```
In [70]: import numpy as np

from nltk.tokenize import WordPunctTokenizer
tokenizer = WordPunctTokenizer()

def question_to_vec(query:str, embeddings:KeyedVectors, tokenizer_engine:Tokenizer):
    """
    query: строка
```

```

embeddings: наше векторное представление
tokenizer_engine: токенайзер
dim: размер любого вектора в нашем представлении

    return: векторное представление для вопроса
"""

if embeddings.vector_size != dim:
    raise ValueError("wrong dim")

tokens = tokenizer_engine.tokenize(query)
summ = np.zeros(dim, dtype=np.float32)
count = 0
for token in tokens:
    if token in embeddings:
        vec = embeddings[token]
        summ += vec
        count = count + 1

if count > 0:
    summ = summ / count

return summ

```

Теперь у нас есть метод для создания векторного представления любого предложения.

Вопрос 2:

- Какая третья (с индексом 2) компонента вектора предложения `I love neural networks` (округлите до 2 знаков после запятой)?

```
In [71]: # Предложение
question = "I love neural networks"

print(f"question_to_vec(question, wv_embeddings, tokenizer)[2]:.2f")
```

-1.29

Оценка близости текстов

Представим, что мы используем идеальные векторные представления слов. Тогда косинусное расстояние между дублирующими предложениями должно быть меньше, чем между случайно взятыми предложениями.

Сгенерируем для каждого из N вопросов R случайных отрицательных примеров и примешаем к ним также настоящие дубликаты. Для каждого вопроса будем ранжировать с помощью нашей модели $R + 1$ примеров и смотреть на позицию дубликата. Мы хотим, чтобы дубликат был первым в ранжированном списке.

Hits@K

Первой простой метрикой будет количество корректных попаданий для какого-то K : $\text{Hits}@K = \frac{1}{N} \sum_{i=1}^N \mathbb{1}_{[\text{rank}_q i \leq K]}$

- $\begin{cases} 1, & x < 0 \\ 0, & x \geq 0 \end{cases}$ - индикаторная функция
- q_i - i -ый вопрос
- q_i' - его дубликат
- $rank(q_i')$ - позиция дубликата в ранжированном списке ближайших предложений для вопроса q_i .

Hits@K измеряет долю вопросов, для которых правильный ответ попал в топ-K позиций среди отранжированных кандидатов.

DCG@K

Второй метрикой будет упрощенная DCG метрика, учитывающая порядок элементов в списке путем домножения релевантности элемента на вес равный обратному логарифму номера позиции: $\text{DCG}@K = \frac{1}{N} \sum_{i=1}^N \frac{1}{\log_2(1+rank(q_i'))} \cdot rank(q_i') \leq K$. С такой метрикой модель штрафуется за большой ранк корректного ответа.

DCG@K измеряет качество ранжирования, учитывая не только факт наличия правильного ответа в топ-K, но и ***его точную позицию***.



Пример оценок

Вычислим описанные выше метрики для игрушечного примера. Пусть

- $N = 1$, $R = 3$
- "Что такое python?" - вопрос q_1
- "Что такое язык python?" - его дубликат q_i'

Пусть модель выдала следующий ранжированный список кандидатов:

1. "Как изучить c++?"
2. "Что такое язык python?"
3. "Хочу учить Java"
4. "Не понимаю Tensorflow"

$$\Rightarrow \text{rank}_{q_i} = 2$$

Вычислим метрику $Hits@K$ для $K = 1, 4$:

- $[K = 1] \text{Hits}@1 = [\text{rank}_{q_1} \leq 1]$

Проверяем условие $\text{rank}_{q_1} \leq 1$: **условие неверно.**

Следовательно, $\text{rank}_{q_1} \leq 1 = 0$.

- $[K = 4] \text{Hits}@4 = [\text{rank}_{q_1} \leq 4] = 1$

Проверяем условие $\text{rank}_{q_1} \leq 4$: **условие верно.**

Вычислим метрику $DCG@K$ для $K = 1, 4$:

- $[K = 1] \text{DCG}@1 = \frac{1}{\log_2(1+2)} \cdot 2 \leq 1 = 0$
- $[K = 4] \text{DCG}@4 = \frac{1}{\log_2(1+2)} \cdot 2 \leq 4 = \frac{1}{\log_2(3)}$

Вопрос 3:

- Вычислите $DCG@10$, если $\text{rank}_{q_i} = 9$ (округлите до одного знака после запятой)

Более сложный пример оценок

Рассмотрим пример с $N > 1$, где $N = 3$ (три вопроса) и для каждого вопроса заданы позиции их дубликатов. Вычислим метрики $Hits@K$ для разных значений K .

- $N = 3$: Три вопроса (q_1, q_2, q_3).
- Для каждого вопроса известна позиция его дубликата ($\text{rank}_{q'_i}$):
 - $\text{rank}_{q'_1} = 2$,
 - $\text{rank}_{q'_2} = 5$,
 - $\text{rank}_{q'_3} = 1$.

Мы будем вычислять $Hits@K$ для $K = 1, 5$.

Для $K = 1$:

Подставим значения: $\text{Hits}@1 = \frac{1}{3} \cdot \left([\text{rank}_{q'_1} \leq 1] + [\text{rank}_{q'_2} \leq 1] + [\text{rank}_{q'_3} \leq 1] \right)$

Проверяем условие $\text{rank}_{q'_i} \leq 1$ для каждого вопроса:

- $\text{rank}_{q'_1} = 2 \rightarrow 2 \not\leq 1 \rightarrow 0$,
- $\text{rank}_{q'_2} = 5 \rightarrow 5 \not\leq 1 \rightarrow 0$,
- $\text{rank}_{q'_3} = 1 \rightarrow 1 \leq 1 \rightarrow 1$.

Сумма: $\frac{1}{3} \cdot (0 + 0 + 1) = \frac{1}{3}$.

$\boxed{\text{Hits@1}} = \frac{1}{3}$.

Для $K = 5$:

Подставим значения: $\text{Hits@5} = \frac{1}{3} \cdot \left(\text{rank}_{q'_1} \leq 5 + \text{rank}_{q'_2} \leq 5 + \text{rank}_{q'_3} \leq 5 \right)$.

Проверяем условие $\text{rank}_{q'_i} \leq 5$ для каждого вопроса:

- $\text{rank}_{q'_1} = 2 \rightarrow 2 \leq 5 \rightarrow 1$,
- $\text{rank}_{q'_2} = 5 \rightarrow 5 \leq 5 \rightarrow 1$,
- $\text{rank}_{q'_3} = 1 \rightarrow 1 \leq 5 \rightarrow 1$.

Сумма: $\frac{1}{3} \cdot (1 + 1 + 1) = 1$.

$\boxed{\text{Hits@5}} = 1$.

Теперь вычислим метрику **DCG@K** для того же примера, где $N = 3$ (три вопроса), и для каждого вопроса известна позиция его дубликата ($\text{rank}_{q'_i}$):

- $\text{rank}_{q'_1} = 2$,
- $\text{rank}_{q'_2} = 5$,
- $\text{rank}_{q'_3} = 1$.

Мы будем вычислять **DCG@K** для $K = 1, 5$.

Для $K = 1$: Подставим значения: $\frac{1}{3} \cdot \left(\frac{1}{\log_2(1 + \text{rank}_{q'_1})} + \frac{1}{\log_2(1 + \text{rank}_{q'_2})} + \frac{1}{\log_2(1 + \text{rank}_{q'_3})} \right)$.

Проверяем условие $\text{rank}_{q'_i} \leq 1$ для каждого вопроса:

- $\text{rank}_{q'_1} = 2 \rightarrow 2 \not\leq 1 \rightarrow 0$,
- $\text{rank}_{q'_2} = 5 \rightarrow 5 \not\leq 1 \rightarrow 0$,
- $\text{rank}_{q'_3} = 1 \rightarrow 1 \leq 1 \rightarrow 1$.

Сумма: $\frac{1}{3} \cdot (0 + 0 + 1) = \frac{1}{3}$.

Для $K = 5$: Подставим значения: $\text{DCG@5} = \frac{1}{3} \cdot \left(\frac{1}{\log_2(1 + \text{rank}_{q'_1})} + \frac{1}{\log_2(1 + \text{rank}_{q'_2})} + \frac{1}{\log_2(1 + \text{rank}_{q'_3})} \right)$

Проверяем условие $\text{rank}_{q'_i} \leq 5$ для каждого вопроса:

- $\text{rank}_{q'_1} = 2 \rightarrow 2 \leq 5 \rightarrow 1$,
- $\text{rank}_{q'_2} = 5 \rightarrow 5 \leq 5 \rightarrow 1$,
- $\text{rank}_{q'_3} = 1 \rightarrow 1 \leq 5 \rightarrow 1$.

Сумма: $\text{DCG@5} = \frac{1}{3} \cdot (0.631 + 0.387 + 1) = \frac{1}{3} \cdot 2.018 \approx 0.673$.

$\boxed{\text{DCG@5} \approx 0.673}$.

Bonus 4:

- Найдите максимум $\text{Hits}@47 - \text{DCG}@1$?

In [72]:

```
import math

rank_q1 = 2
rank_q2 = 5
rank_q3 = 1

ranks = [rank_q1, rank_q2, rank_q3]

hits_k = lambda k_val, ranks_val : sum(1 if r_val <= k_val else 0 for r_val in r
dcg_k = lambda k_val, ranks_val : sum((1 if r_val <= k_val else 0) / (math.log2(1

print(hits_k(47, ranks) - dcg_k(1, ranks)))
```

0.6666666666666667

HITS_COUNT и DCG_SCORE

Каждая функция имеет два аргумента: ranks и k .

ranks является списком, который содержит рейтинги дубликатов (их позиции в ранжированном списке).

К примеру для "Что такое язык python?" $\text{ranks} = [2]$.

In [73]:

```
def hits_count(duplicate_ranks, top_k):
    """
        duplicate_ranks: list индексов дубликатов
        top_k: пороговое значение для ранга
        result: вернуть Hits@k
    """
    # Подсчитываем количество дубликатов, чей ранг <= k
```

```
    return hits_k(top_k, duplicate_ranks)
```

```
In [74]: dup_ranks = [2]

k = 1
hits_value = hits_count(dup_ranks, k)
print(f"Hits@1 = {hits_value}")

k = 4
hits_value = hits_count(dup_ranks, k)
print(f"Hits@4 = {hits_value}")
```

Hits@1 = 0.0
Hits@4 = 1.0

```
In [75]: def dcg_score(duplicate_ranks, top_k):
    """
        duplicate_ranks: list индексов дубликатов
        top_k: пороговое значение для ранга
        result: вернуть DCG@k
    """

    return dcg_k(top_k, duplicate_ranks)
```

```
In [76]: # Пример списка позиций дубликатов
dup_ranks = [2]

# Вычисляем DCG@1
dcg_value = dcg_score(dup_ranks, top_k=1)
print(f"DCG@1 = {dcg_value:.3f}")

# Вычисляем DCG@4
dcg_value = dcg_score(dup_ranks, top_k=4)
print(f"DCG@4 = {dcg_value:.3f}")
```

DCG@1 = 0.000
DCG@4 = 0.631

Протестируем функции. Пусть \$N = 1\$, то есть один эксперимент. Будем искать копию вопроса и оценивать метрики.

```
In [77]: import pandas as pd
```

```
In [78]: copy_answers = ["How does the catch keyword determine the type of exception that

# наши кандидаты
candidates_ranking = [[ "How Can I Make These Links Rotate in PHP",
                        "How does the catch keyword determine the type of exception that",
                        "NSLog array description not memory address",
                        "PECL_HTTP not recognised php ubuntu"],]

# dup_ranks - позиции наших копий, так как эксперимент один, то этот массив длин
dup_ranks = [candidates_ranking[0].index(copy_answers[0]) + 1]

# вычисляем метрику для разных k
print('Ваш ответ HIT:', [hits_count(dup_ranks, k) for k in range(1, 5)])
print('Ваш ответ DCG:', [round(dcg_score(dup_ranks, k), 5) for k in range(1, 5)])
```

Ваш ответ HIT: [0.0, 1.0, 1.0, 1.0]
 Ваш ответ DCG: [0.0, 0.63093, 0.63093, 0.63093]

У вас должно получиться

```
In [79]: # correct_answers - метрика для разных k
correct_answers = pd.DataFrame([[0, 1, 1, 1], [0, 1 / (np.log2(3)), 1 / (np.log2
                           index=['HITS', 'DCG'], columns=range(1,5))
correct_answers
```

	1	2	3	4
HITS	0	1.00000	1.00000	1.00000
DCG	0	0.63093	0.63093	0.63093

Данные

[arxiv link](#)

`train.tsv` - выборка для обучения.

В каждой строке через табуляцию записаны: **<вопрос>**, **<похожий вопрос>**

`validation.tsv` - тестовая выборка.

В каждой строке через табуляцию записаны: **<вопрос>**, **<похожий вопрос>**,
<отрицательный пример 1>, **<отрицательный пример 2>**, ...

```
In [80]: out_path = Path("stackoverflow_similar_questions.zip")

if not out_path.exists():
    import gdown
    import zipfile

    gdown.download(
        id="1QqT4D0EoqJTy7v9VrNCYD-m964XZFR7_",
        output=str(out_path.resolve()),
        quiet=False
    )
    Path("data").mkdir(exist_ok=True)

    with zipfile.ZipFile(out_path, "r") as zf:
        zf.extractall(path=".")
```

Считайте данные.

```
In [81]: def read_corpus(filename):
    data = []
    with open(filename, encoding='utf-8') as file:
        for l in file:
            data.append(l.strip().split('\t'))
    return data
```

Нам понадобиться только файл validation.

```
In [82]: validation_data = read_corpus('./data/validation.tsv')
```

Кол-во строк

```
In [83]: len(validation_data)
```

```
Out[83]: 3760
```

Размер нескольких первых строк

```
In [84]: for i in range(25):
    print(i + 1, len(validation_data[i]))
```

```
1 1001
2 1001
3 1001
4 1001
5 1001
6 1001
7 1001
8 1001
9 1001
10 1001
11 1001
12 1001
13 1001
14 1001
15 1001
16 1001
17 1001
18 1001
19 1001
20 1001
21 1001
22 1001
23 1001
24 1001
25 1001
```

Ранжирование без обучения

Реализуйте функцию ранжирования кандидатов на основе косинусного расстояния.

Функция должна по списку кандидатов вернуть отсортированный список пар

(позиция в исходном списке кандидатов, кандидат). При этом позиция кандидата в полученном списке является его рейтингом (первый - лучший). Например, если исходный список кандидатов был [a, b, c], и самый похожий на исходный вопрос среди них - с, затем а, и в конце б, то функция должна вернуть список [(2, c), (0, a), (1, b)].

```
In [85]: from sklearn.metrics.pairwise import cosine_similarity
```

```
In [86]: from collections import OrderedDict
```

```
def rank_candidates(query:str, candidate_texts:list[str], embeddings:KeyedVector
    """
        query: строка
        candidate_texts: массив строк(кандидатов) [a, b, c]
```

```

embeddings: наше векторное представление
tokenizer_engine: токенайзер
dim: размер любого вектора в нашем представлении

result: пары (начальная позиция, кандидат) [(2, c), (0, a), (1, b)]
"""

if embeddings.vector_size != dim:
    raise ValueError("wrong dim")
query = query.lower() if normalize else query
question_vec = question_to_vec(query, embeddings, tokenizer_engine, dim).res
answers_vec = np.vstack([question_to_vec(c.lower() if normalize else c, embe
similarity = cosine_similarity(question_vec, answers_vec)[0]
similarity_sort = sorted({(idx, candidate_texts[idx]): sim for idx, sim in
result = list(OrderedDict(similarity_sort).keys()))

return result

```

Протестируйте работу функции на примерах ниже. Пусть \$N=2\$, то есть два эксперимента

```
In [87]: questions = ['converting string to list', 'Sending array via Ajax fails']

candidates = [[ 'Convert Google results object (pure js) to Python object', # непр
               'C# create cookie from string and send it',
               'How to use jQuery AJAX for an outside domain?'],

              [ 'Getting all list items of an unordered list in PHP',      # вмоп
                'WPF- How to update the changes in list item of a list',
                'select2 not displaying search results']]
```

Результаты работы с приведением слов к нижнему регистру

```
In [88]: for question, q_candidates in zip(questions, candidates):
            ranks = rank_candidates(question, q_candidates, wv_embeddings, tokenizer
            print(ranks)
            print()

[(1, 'C# create cookie from string and send it'), (0, 'Convert Google results obj
ect (pure js) to Python object'), (2, 'How to use jQuery AJAX for an outside doma
in?')]

[(0, 'Getting all list items of an unordered list in PHP'), (2, 'select2 not disp
laying search results'), (1, 'WPF- How to update the changes in list item of a li
st')]
```

Результаты работы без приведением слов к нижнему регистру

```
In [89]: for question, q_candidates in zip(questions, candidates):
            ranks = rank_candidates(question, q_candidates, wv_embeddings, tokenizer
            print(ranks)
            print()
```

```
[(1, 'C# create cookie from string and send it'), (0, 'Convert Google results object (pure js) to Python object'), (2, 'How to use jQuery AJAX for an outside domain?')]

[(1, 'WPF- How to update the changes in list item of a list'), (0, 'Getting all list items of an unordered list in PHP'), (2, 'select2 not displaying search results')]
```

Для первого эксперимента вы можете полностью сравнить ваши ответы и правильные ответы. Но для второго эксперимента два ответа на кандидаты будут скрыты(*)

In [90]:

```
# должно вывести
results = [[(1, 'C# create cookie from string and send it'),
            (0, 'Convert Google results object (pure js) to Python object'),
            (2, 'How to use jQuery AJAX for an outside domain?')],
           [(0, 'Getting all list items of an unordered list in PHP'),
            (2, 'select2 not displaying search results'),
            (1, 'WPF- How to update the changes in list item of a list')]]
```

Последовательность начальных индексов вы должны получить для эксперимента 1 1, 0, 2.

Вопрос 5:

- Какую последовательность начальных индексов вы получили для эксперимента 2 (перечисление без запятой и пробелов, например, 102 для первого эксперимента?)

Ответ: 021

Теперь мы можем оценить качество нашего метода. Запустите следующие два блока кода для получения результата. Обратите внимание, что вычисление расстояния между векторами занимает некоторое время (примерно 10 минут). Можете взять для validation 1000 примеров.

In [91]:

```
from tqdm.auto import tqdm
from typing import Optional

def build_wv_ranking(data, embeddings: KeyedVectors, tokenizer_engine: Tokenizer,
                     wv_ranking_result = []):
    for idx, item in enumerate(tqdm(data)):
        if limit is not None and idx == limit:
            break
        query, *examples = item
        ranks_result = rank_candidates(query, examples, embeddings, tokenizer_en
wv_ranking_result.append([rank[0] for rank in ranks_result].index(0) + 1)

    return wv_ranking_result
```

In [92]:

```
wv_ranking = build_wv_ranking(validation_data, wv_embeddings, tokenizer)
for k in tqdm([1, 5, 10, 100, 500, 1000]):
    print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg_score(wv_ranking, k), k, hi
```

0%	0/3760 [00:00<?, ?it/s]
0%	0/6 [00:00<?, ?it/s]
DCG@ 1:	0.408 Hits@ 1: 0.408
DCG@ 5:	0.498 Hits@ 5: 0.578
DCG@ 10:	0.522 Hits@ 10: 0.651
DCG@ 100:	0.567 Hits@ 100: 0.868
DCG@ 500:	0.580 Hits@ 500: 0.975
DCG@1000:	0.583 Hits@1000: 1.000

Из формул выше можно понять, что

- $\text{Hits}@K$ **монотонно неубывающая функция** K , которая стремится к 1 при $K \rightarrow \infty$.
- $\text{DCG}@K$ **монотонно неубывающая функция** K , но рост замедляется с увеличением K из-за убывания веса $\frac{1}{\log_2(1 + \text{rank}_q)}$.

Эмбеддинги, обученные на корпусе похожих вопросов

In [93]: `train_data = read_corpus('./data/train.tsv')`

Улучшите качество модели.

Склейм вопросы в пары и обучим на них модель Word2Vec из gensim. Выберите размер window. Объясните свой выбор.

Рассмотрим подробнее данное склеивание.

1. Каждая строка из `train_data` разбивается на вопрос (`question`) и список кандидатов.
2. Для каждого кандидата вопрос склеивается с ним в одну строку.
3. Склеенная строка (`combined_text`) токенизируется, и полученный список токенов добавляется в общий корпус (`corpus`).

Пример

Вопрос: "What is Python?"

Кандидаты: ["Python is a programming language", "Java is another language"]

Склейные строки:

"What is Python? Python is a programming language"
"What is Python? Java is another language"

Токенизированные списки:

```
['what', 'is', 'python', 'python', 'is', 'a', 'programming',
'language']
['what', 'is', 'python', 'java', 'is', 'another',
'language']
```

```
In [94]: train_data[111258]
```

```
Out[94]: ['Determine if the device is a smartphone or tablet?',  
 'Change imageView params in all cards together']
```

```
In [95]: # Создаем общий корпус текстов  
def build_corpus(data:list[list[str]], tokenizer_engine:Tokenizer, normalize:bool)  
    corpus_result = []  
    for item in data:  
        if len(item) <= 1:  
            continue  
        query = item[0]  
        for examples in item[1:]:  
            concat_text = f"{query} {examples}"  
            corpus_result.append(tokenizer_engine.tokenize(concat_text.lower()))  
    return corpus_result
```

```
In [96]: corpus = build_corpus(train_data, tokenizer)
```

```
In [97]: max_tokens = -1  
for t in corpus:  
    max_tokens = max(max_tokens, len(t))  
print(f"max tokens: {max_tokens}")
```

max tokens: 105

Пойщем разные варианты для windows и current_count

```
In [98]: # from gensim.models import Word2Vec  
# target_params = []  
# for current_windows in ( 10, 105, 150, 180):  
#     for current_count in (2,3,5,10):  
#         embeddings_trained = Word2Vec(  
#             sentences=corpus,  
#             vector_size=200,  
#             window=current_windows,  
#             min_count=current_count,  
#             workers=32  
#         )  
#         kv = embeddings_trained.wv  
#         ranks = build_wv_ranking(validation_data, kv, tokenizer, limit=1000, n  
#         hits10 = hits_count(ranks, 10)  
#         dcg10 = dcg_score(ranks, 10)  
#         target_params.append({  
#             "window": current_windows,  
#             "min_count": current_count,  
#             "Hits@10": hits10,  
#             "DCG@10": dcg10  
#         })  
# target_params.sort(key=lambda x: (x["DCG@10"], x["Hits@10"])), reverse=True)  
# print(target_params)  
#
```

```
[{'window': 180, 'min_count': 10, 'Hits@10': 0.594, 'DCG@10':  
 0.4836081202584917}, {'window': 180, 'min_count': 5, 'Hits@10': 0.585,  
 'DCG@10': 0.4767133140367386}, {'window': 150, 'min_count': 10,  
 'Hits@10': 0.584, 'DCG@10': 0.4759279272314121}, {'window': 150,
```

```
'min_count': 5, 'Hits@10': 0.578, 'DCG@10': 0.474757934630991},
{'window': 180, 'min_count': 3, 'Hits@10': 0.575, 'DCG@10':
0.4738912073715378}, {'window': 105, 'min_count': 10, 'Hits@10': 0.582,
'DCG@10': 0.47368553605576336}, {'window': 105, 'min_count': 5,
'Hits@10': 0.581, 'DCG@10': 0.47293118799157347}, {'window': 150,
'min_count': 2, 'Hits@10': 0.576, 'DCG@10': 0.4728387728645323},
{'window': 150, 'min_count': 3, 'Hits@10': 0.579, 'DCG@10':
0.47245360173821804}, {'window': 105, 'min_count': 3, 'Hits@10': 0.578,
'DCG@10': 0.4712054486877369}, {'window': 180, 'min_count': 2,
'Hits@10': 0.576, 'DCG@10': 0.46940530029760863}, {'window': 105,
'min_count': 2, 'Hits@10': 0.573, 'DCG@10': 0.4652144479921041},
{'window': 10, 'min_count': 5, 'Hits@10': 0.523, 'DCG@10':
0.41083477592221823}, {'window': 10, 'min_count': 10, 'Hits@10': 0.521,
'DCG@10': 0.4101784880176292}, {'window': 10, 'min_count': 2,
'Hits@10': 0.522, 'DCG@10': 0.4089606516853269}, {'window': 10,
'min_count': 3, 'Hits@10': 0.524, 'DCG@10': 0.4087087391619807}]
```

Видно что если выставить аномально высокие значения для window качество немного растет (с DCG@10 с 0.40 до 0.48). Гипотеза почему так происходит:

При расчете build_wv_ranking не учитывается порядок слов, эмбеддинг для предложения считается как сумма эмбеддинг слов. В итоговый вектор все слова предложения вносят одинаковый вклад.

При обучение, с большим значением windows, взаимосвязи ищутся между всеми словами в предложение. Учитываются взаимосвязи между словами, которые стоят в начале и конце предложения.

Т.е. мы подталкиваем Word2Vec делать такие эмбеддинги, которые работают лучше, для случая, когда эмбеддинг предложения, это сумма эмбеддинг слов.

min_count - отбрасывает токены. Для большого значения window отбрасывания токенов, помогает лучше "усреднить" результат, т.е. также адаптирует Word2Vec создавать ембединги, которые лучше подходят для метода когда ембединг предложения = сумме ембедингов слов.

Эти улучшения незначительны, но требуют больше ресурсов. Поэтому в практических целях буду использовать window=10 и min_count=5. В целом от способа, когда целевой эмбеддинг предложения, это сумма эмбеддингов слов, нужно уходить к способу, когда слова содержащие "смысл" предложения вносят вклад в эмбеддинг предложения больше, чем слова с "низким" смыслом. При таком способе, большие значения window только бы вредили, т.к. выстраивали взаимосвязи между словами, которые по смыслу не связаны.

```
In [99]: from gensim.models import Word2Vec

embeddings_trained = Word2Vec(
    sentences=corpus,           # Корпус токенизованных текстов
    vector_size=200,             # Размерность векторов
```

```

        window=10,           # Размер окна контекста
        min_count=5,         # Минимальная частота слов
        workers=8            # Количество потоков
    ).wv

```

In [100...]:

```
wv_ranking = build_wv_ranking(validation_data, embeddings_trained, tokenizer)
```

0% | 0/3760 [00:00<?, ?it/s]

In [101...]:

```
for k in tqdm([1, 5, 10, 100, 500, 1000]):
    print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg_score(wv_ranking, k), k, hits_at_k(wv_ranking, k, k)))
```

k	DCG@k	Hits@k
1	0.314	0.314
5	0.392	0.461
10	0.414	0.528
100	0.465	0.779
500	0.486	0.948
1000	0.492	1.000

Замечание:

Решить эту задачу с помощью обучения полноценной нейронной сети будет вам предложено, как часть задания в одной из домашних работ по теме "Диалоговые системы".

Напишите свой вывод о полученных результатах.

- Какой принцип токенизации даёт качество лучше и почему?
- Помогает ли нормализация слов?
- Какие эмбеддинги лучше справляются с задачей и почему?
- Почему получилось плохое качество решения задачи?
- Предложите свой подход к решению задачи.

Вывод:

Какой принцип токенизации даёт качество лучше и почему

Сделаем проверку для трех токенайзеров

```

In [102...]: import re
import spacy
from spacy.cli import download

try:
    nlp = spacy.load("en_core_web_sm")
except OSError:
    download("en_core_web_sm")
    nlp = spacy.load("en_core_web_sm")

class TokenizerRegexp(Tokenizer):
    def __init__(self):

```

```

    pass
def tokenize(self, text):
    return re.findall(r'\w+', text)

class TokenizerSpacy(Tokenizer):
    def __init__(self):
        pass
    def tokenize(self, text):
        doc = nlp.make_doc(text)
        return [txt.text for txt in doc]

tokenizers = {
    "Корректная обработка - апострофов и сокращений(spaCy)": TokenizerSpacy(),
    "Последовательности из букв и цифр, отсеивается пунктуация и апострофы(регулярка)": WordPunctTokenizer(),
    "Дробление с учетом знаков пунктуации(WordPunct)": WordPunctTokenizer(),
}

for tokenizer_name, current_tokenizer in tokenizers.items():
    wv_ranking = build_wv_ranking(validation_data, wv_embeddings, current_tokenizer)
    print(tokenizer_name)
    for k in tqdm([1, 5, 10, 100, 500, 1000]):

        print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg_score(wv_ranking, k), k))
        print()

```

0% | 0/3760 [00:00<?, ?it/s]

Корректная обработка - апострофов и сокращений(spaCy)

0%	0/6 [00:00<?, ?it/s]
DCG@ 1:	0.406 Hits@ 1: 0.406
DCG@ 5:	0.496 Hits@ 5: 0.575
DCG@ 10:	0.518 Hits@ 10: 0.643
DCG@ 100:	0.563 Hits@ 100: 0.864
DCG@ 500:	0.577 Hits@ 500: 0.971
DCG@1000:	0.580 Hits@1000: 1.000

0% | 0/3760 [00:00<?, ?it/s]

Последовательности из букв и цифр, отсеивается пунктуация и апострофы(регулярка)

0%	0/6 [00:00<?, ?it/s]
DCG@ 1:	0.415 Hits@ 1: 0.415
DCG@ 5:	0.502 Hits@ 5: 0.582
DCG@ 10:	0.525 Hits@ 10: 0.651
DCG@ 100:	0.570 Hits@ 100: 0.874
DCG@ 500:	0.583 Hits@ 500: 0.973
DCG@1000:	0.586 Hits@1000: 1.000

0% | 0/3760 [00:00<?, ?it/s]

Дробление с учетом знаков пунктуации(WordPunct)

0%	0/6 [00:00<?, ?it/s]
DCG@ 1:	0.410 Hits@ 1: 0.410
DCG@ 5:	0.500 Hits@ 5: 0.580
DCG@ 10:	0.521 Hits@ 10: 0.645
DCG@ 100:	0.568 Hits@ 100: 0.875
DCG@ 500:	0.581 Hits@ 500: 0.973
DCG@1000:	0.583 Hits@1000: 1.000

Вопрос: Какой принцип токенизации даёт качество лучше и почему? Ответ: Зависит от задачи. Для данной задачи лучшее качество показал токенайзер на простой

регулярке. Почему так:

- С большой вероятностью способ разбиения по регулярке максимально совпадает с способом токенезации который использовался для эмбедингов
- В контексте данных с SO пунктуации и апострофы - являются данными создающим шум. Их отбрасывание помогает получать более "стабильные" ветокра эмбедингов

Помогает ли нормализация слов

Возьмем разные способы нормализации слов и проверим на regexp токенайзере

```
In [103...]: from typing import Callable
from nltk.corpus import stopwords
import nltk
from nltk.stem.snowball import SnowballStemmer

nltk.download('stopwords')

class TokenizerNormalize(Tokenizer):
    def __init__(self, normalizer_func:Callable[[str], str]):
        self.normalizer_func = normalizer_func
    def tokenize(self, text)-> list[str]:
        return re.findall(r'\w+', self.normalizer_func(text))

class TokenizerNormalizeStopWords(Tokenizer):
    def __init__(self):
        self.stop = set(stopwords.words("english"))
    def tokenize(self, text)-> list[str]:
        tokens = re.findall(r'\w+', text)
        return [txt for txt in tokens if txt not in self.stop]

class TokenizerRegexList(Tokenizer):
    def tokenize(self, text: str) -> list[str]:
        return re.findall(r"[A-Za-z0-9\+\#\.\.]+", text)

class TokenizerLemmatizer(Tokenizer):
    def __init__(self):
        self.stem = SnowballStemmer("english")
    def tokenize(self, text: str) -> list[str]:
        return [self.stem.stem(txt) for txt in text.split()]

tokenizers = {
    "Лемматизация с SnowballStemmer": TokenizerLemmatizer(),
    "Без нормализации": TokenizerNormalize(lambda x: x),
    "Приведение текста к нижнему регистру": TokenizerNormalize(lambda x: x.lower),
    "Удаление стоп слов": TokenizerNormalizeStopWords(),
    "Удаление пунктуации и части символов. Оставляем символы которые встречаются"
}

for tokenizer_name, current_tokenizer in tokenizers.items():
    wv_ranking = build_wv_ranking(validation_data, wv_embeddings, current_tokenizer)
    print(tokenizer_name)
    for k in tqdm([1, 5, 10, 100, 500, 1000]):
```

```

        print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg_score(wv_ranking, k), k
print()

```

[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\and-r\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!

0% | 0/3760 [00:00<?, ?it/s]

Лемматизация с SnowballStemmer

	DCG@	Hits@	
0%	1: 0.268	1: 0.268	
DCG@	5: 0.343	Hits@ 5: 0.408	
DCG@	10: 0.365	Hits@ 10: 0.478	
DCG@	100: 0.415	Hits@ 100: 0.723	
DCG@	500: 0.440	Hits@ 500: 0.919	
DCG@1000:	0.449	Hits@1000: 1.000	

0% | 0/3760 [00:00<?, ?it/s]

Без нормализации

	DCG@	Hits@	
0%	1: 0.415	1: 0.415	
DCG@	5: 0.502	Hits@ 5: 0.582	
DCG@	10: 0.525	Hits@ 10: 0.651	
DCG@	100: 0.570	Hits@ 100: 0.874	
DCG@	500: 0.583	Hits@ 500: 0.973	
DCG@1000:	0.586	Hits@1000: 1.000	

0% | 0/3760 [00:00<?, ?it/s]

Приведение текста к нижнему регистру

	DCG@	Hits@	
0%	1: 0.415	1: 0.415	
DCG@	5: 0.502	Hits@ 5: 0.582	
DCG@	10: 0.525	Hits@ 10: 0.651	
DCG@	100: 0.570	Hits@ 100: 0.874	
DCG@	500: 0.583	Hits@ 500: 0.973	
DCG@1000:	0.586	Hits@1000: 1.000	

0% | 0/3760 [00:00<?, ?it/s]

Удаление стоп слов

	DCG@	Hits@	
0%	1: 0.416	1: 0.416	
DCG@	5: 0.503	Hits@ 5: 0.582	
DCG@	10: 0.525	Hits@ 10: 0.651	
DCG@	100: 0.571	Hits@ 100: 0.876	
DCG@	500: 0.584	Hits@ 500: 0.973	
DCG@1000:	0.587	Hits@1000: 1.000	

0% | 0/3760 [00:00<?, ?it/s]

Удаление пунктуации и части символов. Оставляем символы которые встречаются в IT

	DCG@	Hits@	
0%	1: 0.424	1: 0.424	
DCG@	5: 0.514	Hits@ 5: 0.595	
DCG@	10: 0.534	Hits@ 10: 0.657	
DCG@	100: 0.579	Hits@ 100: 0.876	
DCG@	500: 0.592	Hits@ 500: 0.976	
DCG@1000:	0.594	Hits@1000: 1.000	

Вопрос: Помогает ли нормализация слов? Ответ: Незначительно помогает "умное" удаление стоп слов.

По остальным результатам:

- лематизация - делает сильно хуже, эмбеддинги явно предобучались без лематизации
- без нормализации/приведение к нижнему регистру - по факту одинаково. слов в верхнем регистре мало, они не вносят существенного вклада.
- удаление стоп слов, чуть улучшает. т.к меньше шума
- удаление стоп слов, но оставление символов которые специфичны для языков программирования. самый хороший результат. но в целом не сильно отличающийся от варианта без нормализации

Какие эмбеддинги лучше справляются с задачей и почему?

С задачей лучше справляются предобученные эмбеддинги

DCG@ 1: 0.408		Hits@ 1: 0.408
DCG@ 5: 0.498		Hits@ 5: 0.578
DCG@ 10: 0.522		Hits@ 10: 0.651
DCG@ 100: 0.567		Hits@ 100: 0.868
DCG@ 500: 0.580		Hits@ 500: 0.975
DCG@1000: 0.583		Hits@1000: 1.000

чем эмбеддинг обученный на нашей выборке

DCG@ 1: 0.315		Hits@ 1: 0.315
DCG@ 5: 0.395		Hits@ 5: 0.469
DCG@ 10: 0.416		Hits@ 10: 0.532
DCG@ 100: 0.466		Hits@ 100: 0.777
DCG@ 500: 0.488		Hits@ 500: 0.946
DCG@1000: 0.493		Hits@1000: 1.000

Возможные причины:

- предобученный эмбеддинг учились на данных большего объема
- лучшая предобработка текста для предобученных эмбеддингов

Почему получилось плохое качество решения задачи?

- Основная причина плохого результата в способе получения эмбеддинг для предложения: это сумма векторов слов. Слова в разном порядке дадут один и тот же вектор. При этом смысл такого предложения может быть координально разным.

```
In [104...]:  
if np.array_equal(  
    question_to_vec("Dog bites man.", wv_embeddings, tokenizer),  
    question_to_vec("Man bites dog.", wv_embeddings, tokenizer)  
):  
    print("Equal")
```

Предложите свой подход к решению задачи.

- Улучшить фильтрацию по стоп словам. Лучше учитывать слова специфичные для домена
- Если возможен фойнтюн для KeyedVectors то пробовать дообучить модель с маленькой скоростью дообучения, на примерах где есть слова специфичные для домена StackOverflow
- Возможно приспособить tf-idf, что бы создать пары вопрос/ответ только с значимыми словами?