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 ранжировния, поэтому введем математическую формулировку

Задача ранжирования(Learning to Rank)

- X - множество объектов
- $X^l = \{x_1, x_2, \ldots, x_l\}$ - 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^l с индексами i и j

  Задача:
  построить ранжирующую функцию $a : X \to R$ такую, что

$$i \prec j \Rightarrow a(x_i) < a(x_j)$$

Embeddings

Будем использовать предобученные векторные представления слов на постах Stack Overflow. A word2vec model trained on Stack Overflow posts

```
!wget https://zenodo.org/record/1199620/files/SO_vectors_200.bin?
download=1

--2025-09-30 13:41:03--
https://zenodo.org/record/1199620/files/SO_vectors_200.bin?download=1
Resolving zenodo.org (zenodo.org)... 188.185.48.194, 188.185.43.25,
188.185.45.92, ...
Connecting to zenodo.org (zenodo.org)|188.185.48.194|:443...
connected.
```

```
HTTP request sent, awaiting response... 301 MOVED PERMANENTLY
Location: /records/1199620/files/SO_vectors_200.bin [following]
--2025-09-30 13:41:03--
https://zenodo.org/records/1199620/files/SO_vectors_200.bin
Reusing existing connection to zenodo.org:443.
HTTP request sent, awaiting response... 200 OK
Length: 1453905423 (1.4G) [application/octet-stream]
Saving to: ‘SO_vectors_200.bin?download=1’

SO_vectors_200.bin? 100%[===================>]   1.35G  1.70MB/s    in
15m 40s

2025-09-30 13:56:44 (1.47 MB/s) - ‘SO_vectors_200.bin?download=1’
saved [1453905423/1453905423]


!pip install gensim

Requirement already satisfied: gensim in
/opt/anaconda3/lib/python3.12/site-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in
/opt/anaconda3/lib/python3.12/site-packages (from gensim) (1.26.4)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in
/opt/anaconda3/lib/python3.12/site-packages (from gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in
/opt/anaconda3/lib/python3.12/site-packages (from gensim) (5.2.1)

from gensim.models.keyedvectors import KeyedVectors
wv_embeddings =
KeyedVectors.load_word2vec_format("SO_vectors_200.bin_download=1",
binary=True)
```

Как пользоваться этими векторами?

Посмотрим на примере одного слова, что из себя представляет embedding

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

float32 (200,)

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?

```
# method most_simmilar
word = "dog"
top5 = wv_embeddings.most_similar(word, topn=5)
print(top5)
```

```
[('animal', 0.8564179539680481), ('dogs', 0.7880867123603821),
('mammal', 0.7623804211616516), ('cats', 0.7621253728866577),
('animals', 0.7607938647270203)]
```

```
import numpy as np
from numpy.linalg import norm
word1, word2 = "dog", "cat"

# мера сходства
sim = wv_embeddings.similarity(word1, word2)
print(f"Cosine similarity({word1}, {word2}) = {sim:.4f}")

# Берём вектор первого слова
vec = wv_embeddings[word1]

# Считаем косинусное сходство word1 со всеми словами одним векторным
умножением
sims = np.dot(wv_embeddings.vectors, vec) / (
    norm(wv_embeddings.vectors, axis=1) * norm(vec)
)
print(sims)

# Сортируем индексы слов по убыванию близости
ranking = np.argsort(-sims)
print(ranking)

# Находим позицию word2 в этом рейтинге
rank_position = np.where(ranking == wv_embeddings.key_to_index[word2])
[0][0] + 1

print(f"Позиция {word2} в списке ближайших к {word1}:
{rank_position}")
```

```
Cosine similarity(dog, cat) = 0.6852
[0.07585394 0.21143065 0.00233965 ... 0.16834745 0.03601661
0.05869402]
[   3880    4323    9942 ... 1191840    2882    4971]
Позиция cat в списке ближайших к dog: 27
```

Ответ: 'cat' не входит в топ-5 ближайших слов к 'dog'. 'cat' находится на 27 месте

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.

```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/nikitacernov/nltk_data...
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /Users/nikitacernov/nltk_data...
```

```
True
```

```
import numpy as np
import re
from nltk.tokenize import WordPunctTokenizer
from nltk.stem import WordNetLemmatizer

class MyTokenizer:
    def __init__(self):
        self.tokenizer = WordPunctTokenizer()
        self.lemmatizer = WordNetLemmatizer()
    def tokenize(self, text, tokenizer, lemm=True):
      if tokenizer == 're':
        tokens = re.findall('\w+', text.lower())
      elif tokenizer == 'WordPunctTokenizer':
        tokens = self.tokenizer.tokenize(text.lower())
      if lemm:
         tokens = [self.lemmatizer.lemmatize(t) for t in tokens]
      return tokens

tokenizer = MyTokenizer()
```

```
<>:12: SyntaxWarning: invalid escape sequence '\w'
<>:12: SyntaxWarning: invalid escape sequence '\w'
/var/folders/yr/s_88h1jx275_s5hfylyqqmxc0000gn/T/ipykernel_36433/39590
54116.py:12: SyntaxWarning: invalid escape sequence '\w'
  tokens = re.findall('\w+', text.lower())
```

```
ss = 'I love neural networks'
ss = 'размер любого вектора в нашем представлении'
tokens = tokenizer.tokenize(ss, 'WordPunctTokenizer')
tokens
```

```
['размер', 'любого', 'вектора', 'в', 'нашем', 'представлении']
```

```
vectors = []
count = 0
for word in tokens:
  if word in wv_embeddings:
```

```
      vectors.append(wv_embeddings[word])
      count+=1

if count == 0:
  mean_vec = np.zeros(200)
else:
  mean_vec = np.mean(vectors, axis=0)  # shape будет (200,)

print(mean_vec.shape, mean_vec)
```

```
(200,) [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0.
 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
def question_to_vec(question, embeddings, tokenizer, dim=200):
    """
        question: строка
        embeddings: наше векторное представление
        dim: размер любого вектора в нашем представлении

        return: векторное представление для вопроса
    """

    tokens = tokenizer.tokenize(question, 're')
    vectors = []
    count = 0

    for word in tokens:
      if word in embeddings:
          vectors.append(embeddings[word])
          count+=1

    if count == 0:
      mean_vec = np.zeros(dim)
    else:
      mean_vec = np.mean(vectors, axis=0)
```

```
    return mean_vec
```

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)?

```
# Предложение
question = "I love neural networks"

question_to_vec(question, wv_embeddings, tokenizer)[2].round(2)
```

```
-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:

$$\text{Hits@K} = \frac{1}{N}\sum_{i=1}^{N} \square\left[rank_i^{'} \le K\right),$$

- $\left[x<0\right) \equiv \begin{pmatrix} 1, & x<0 \\ 0, & x \ge 0 \end{pmatrix}$ - индикаторная функция
- q_i - i-ый вопрос
- $q_i^{'}$ - его дубликат
- $rank_i^{'}$ - 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::

$$\text{DCG@K} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\log_2(1+rank_i')}\cdot[rank_i' \le K],$$

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"

$\Rightarrow rank_i' = 2$

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

- [K = 1] $\text{Hits@1} = [rank_i' \le 1)$

Проверяем условие $ \text{rank}_{q'_1} \leq 1 $: ***условие неверно***.

Следовательно, $[\text{rank}_{q'_1} \le 1) = 0$.

- [K = 4] $\text{Hits@4} = [rank_i' \le 4) = 1$

Проверяем условие $ \text{rank}_{q'_1} \leq 4 $: ***условие верно***.

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

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

Вопрос 3:

- Вычислите `DCG@10`, если $rank_i' = 9$ (округлите до одного знака после запятой)

Ответ:

- [K = 10] $\text{DCG@10} = \frac{1}{\log_2(1+9)} \cdot [9 \leq 10] = \frac{1}{\log_2 10} = 0.3$

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

Сумма:

$$\text{Hits@1} = \frac{1}{3} \cdot (0 + 0 + 1) = \frac{1}{3}.$$

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

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

Сумма:

$$\text{Hits@5} = \frac{1}{3} \cdot (1 + 1 + 1) = 1.$$

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

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:

$$\text{DCG@1} = \frac{1}{3} \cdot \left(\frac{1}{\log_2(1 + \text{rank}_{q'_1})} \cdot [\text{rank}_{q'_1} \leq 1] + \frac{1}{\log_2(1 + \text{rank}_{q'_2})} \cdot [\text{rank}_{q'_2} \leq 1] + \frac{1}{\log_2(1 + \text{rank}_{q'_3})} \cdot [\text{rank}_{q'_3} \leq 1] \right).$$

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

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

Сумма:

$$\text{DCG@1} = \frac{1}{3} \cdot (0 + 0 + 1) = \frac{1}{3}.$$

$$\text{DCG@1} = \frac{1}{3}.$$

Для $ K = 5 $: Подставим значения:

$$\text{DCG@5} = \frac{1}{3} \cdot \left(\frac{1}{\log_2(1 + \text{rank}_{q'_1})} \cdot [\text{rank}_{q'_1} \leq 5] + \frac{1}{\log_2(1 + \text{rank}_{q'_2})} \cdot [\text{rank}_{q'_2} \leq 5] + \frac{1}{\log_2(1 + \text{rank}_{q'_3})} \cdot [\text{rank}_{q'_3} \leq 5] \right).$$

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

- $ \text{rank}_{q'_1} = 2 $ → $ 2 \leq 5 $ → $ 1 $,

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

Сумма:

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

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

Вопрос 4:

- Найдите максимум `Hits@47 - DCG@1`?

Ответ:

- `Hits@47`
  - Максимум будет 1, если документ стоит на позиции от 1 до 47.
- `DCG@1`
  - Минимум будет 0, если документ стоит на позиции больше, чем 1.
- Максимум `Hits@47 - DCG@1`
  - Пример ситуации для максимума: релевантный документ есть в топ-47, но он стоит, скажем, на позиции 10 (первый результат нерелевантен). Тогда Hits@47=1, DCG@1=0, разница = 1.

HITS_COUNT и DCG_SCORE

Каждая функция имеет два аргумента: dup_ranks и k.

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

К примеру для "Что такое язык python?" $dup_ranks = [2]$.

```
def hits_count(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        k: пороговое значение для ранга
        result: вернуть Hits@k
    """
    # Подсчитываем количество дубликатов, чей ранг <= k
    hits_value = 0
    for dupl_pos in dup_ranks:
      if dupl_pos <= k:
        hits_value+=1
    return hits_value/len(dup_ranks)

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
```

```
import math

def dcg_score(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        k: пороговое значение для ранга
        result: вернуть DCG@k
    """
    # Вычисляем сумму для всех релевантных дубликатов
    sum_value = 0
    for dupl_pos in dup_ranks:
      if dupl_pos <= k:
        sum_value += 1 / math.log2(dupl_pos + 1)

    return sum_value/len(dup_ranks)

# Пример списка позиций дубликатов
dup_ranks = [2]

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

# Вычисляем DCG@4
dcg_value = dcg_score(dup_ranks, k=4)
print(f"DCG@10 = {dcg_value:.3f}")
```

```
DCG@1 = 0.000
DCG@10 = 0.631
```

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

```
import pandas as pd

copy_answers = ["How does the catch keyword determine the type of
exception that was thrown",]

# наши кандидаты
candidates_ranking = [["How Can I Make These Links Rotate in PHP",
                       "How does the catch keyword determine the type
of exception that was thrown",
```

```
                        "NSLog array description not memory address",
                        "PECL_HTTP not recognised php ubuntu"],]

# dup_ranks — позиции наших копий, так как эксперимент один, то этот
массив длины 1
dup_ranks = [2]

# 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

```
# correct_answers - метрика для разных k
correct_answers = pd.DataFrame([[0, 1, 1, 1], [0, 1 / (np.log2(3)), 1
/ (np.log2(3)), 1 / (np.log2(3))]],
                               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: **,**

`validation.tsv` - тестовая выборка. В каждой строке через табуляцию записаны: **,, <отрицательный пример 1>, <отрицательный пример 2>, ...**

```
!unzip stackoverflow_similar_questions.zip
```

```
Archive:  stackoverflow_similar_questions.zip
   creating: data/
  inflating: data/.DS_Store
   creating: __MACOSX/
   creating: __MACOSX/data/
  inflating: __MACOSX/data/._.DS_Store
  inflating: data/train.tsv
  inflating: data/validation.tsv
```

Считайте данные.

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

Нам понадобиться только файл validation.

```
validation_data = read_corpus('./data/validation.tsv')
```

Кол-во строк

```
len(validation_data)
```

```
3760
```

Размер нескольких первых строк

```
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 - c, затем a, и в конце b, то функция должна вернуть список **[(2, c), (0, a), (1, b)]**.

```
from sklearn.metrics.pairwise import cosine_similarity
from copy import deepcopy

def rank_candidates(question, candidates, embeddings, tokenizer,
dim=200):
    """
        question: строка
        candidates: массив строк(кандидатов) [a, b, c]
        result: пары (начальная позиция, кандидат) [(2, c), (0, a),
(1, b)]
    """

    cand_vec = []
    for i in range(len(candidates)):
      cand_vec.append(question_to_vec(candidates[i], embeddings,
tokenizer))

    quest_vec = question_to_vec(question, embeddings, tokenizer)
    sim_score = [cosine_similarity(quest_vec.reshape(1, -1),
cand_vec[i].reshape(1, -1))[0][0] for i in range(len(cand_vec))]
    # sorted_pairs = sorted(zip(sim_score, candidates), key=lambda x:
x[0], reverse=True) # x[0] «сортируй по первому элементу кортежа (т.е.
по score)».
    indexed = list(enumerate(zip(sim_score, candidates)))
    sorted_pairs = sorted(indexed, key=lambda x: x[1][0],
reverse=True)

    return sorted_pairs

# cand_vec = []
# for i in range(len(candidates[0])):
#  cand_vec.append(question_to_vec(candidates[0][i], wv_embeddings,
tokenizer))

# quest_vec = question_to_vec(questions[0], wv_embeddings, tokenizer)

# sim_score = [cosine_similarity(quest_vec.reshape(1, -1),
cand_vec[i].reshape(1, -1))[0][0] for i in range(len(cand_vec))]

# sorted_pairs = sorted(zip(sim_score, candidates[0]), key=lambda x:
x[0], reverse=True) # x[0] «сортируй по первому элементу кортежа (т.е.
```

```
по score)».
# indexed = list(enumerate(zip(sim_score, candidates[0])))
# print(sorted_pairs)
# print(indexed)
# # сортируем по score (берём элемент [1][0])
# sorted_pairs = sorted(indexed, key=lambda x: x[1][0], reverse=True)
# print(sorted_pairs)
```

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

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

candidates = [['Convert Google results object (pure js) to Python
object', # первый эксперимент
               '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',
# второй эксперимент
               'WPF- How to update the changes in list item of a
list',
               'select2 not displaying search results']]

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

```
converting string to list
[(1, (0.5922089, 'C# create cookie from string and send it')), (0,
(0.5414996, 'Convert Google results object (pure js) to Python
object')), (2, (0.097037084, 'How to use jQuery AJAX for an outside
domain?'))]

Sending array via Ajax fails
[(0, (0.4613238, 'Getting all list items of an unordered list in
PHP')), (2, (0.32301265, 'select2 not displaying search results')),
(1, (0.3005938, 'WPF- How to update the changes in list item of a
list'))]
```

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

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

  Cell In[27], line 5
    [(*, 'Getting all list items of an unordered list in PHP'), #скрыт
       ^
SyntaxError: Invalid star expression
```

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.

```
from tqdm.notebook import tqdm

wv_ranking = []
max_validation_examples = 1000
for i, line in enumerate(tqdm(validation_data)):
    # if i == max_validation_examples:
    #     break
    q, *ex = line
    ranks = rank_candidates(q, ex, wv_embeddings, tokenizer)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

{"model_id":"2dd1c0dc55da4f889d2445211bd3c032","version_major":2,"version_minor":0}

```
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_count(wv_ranking, k)))
```

{"model_id":"4a6fb3bc64c04be7bdfc778ac33090b7","version_major":2,"version_minor":0}

```
DCG@   1: 0.412 | Hits@   1: 0.412
DCG@   5: 0.500 | Hits@   5: 0.579
DCG@  10: 0.525 | Hits@  10: 0.656
DCG@ 100: 0.570 | Hits@ 100: 0.874
DCG@ 500: 0.584 | Hits@ 500: 0.976
DCG@1000: 0.586 | Hits@1000: 1.000
```

Из формул выше можно понять, что

- $ \text{Hits@K} $ **монотонно неубывающая функция** $ K $, которая стремится к 1 при $ K \to \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'_i})} $.

Эмбеддинги, обученные на корпусе похожих вопросов

```
train_data = read_corpus('./data/train.tsv')
```

Улучшите качество модели.Склеим 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"]
Склеенные 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']
```

```
train_data[8]
```

```
['Adding Prototype to JavaScript Object Literal',
 'How does JavaScript .prototype work?',
 'Javascript not setting property of undefined in prototyped object']
```

Без лемматизации

```
# Создаем общий корпус текстов
corpus = []

for item in tqdm(train_data):
    query, answer = item[0], item[1:]
    # Склейка
    pairs = [tokenizer.tokenize(f'{query} {c}', 're', False) for c in
answer]
    corpus.extend(pairs)
    # break
len(corpus)
```

{"model_id":"be1add548f0c43d7b45ccb2c3d68cca7","version_major":2,"vers
ion_minor":0}

```
1256483
```

```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(
    sentences=corpus,        # Корпус токенизированных текстов
    vector_size=200,         # Размерность векторов
    window=3,                # Размер окна контекста
    min_count=3,             # Минимальная частота слов
    workers=4                # Количество потоков
).wv

wv_ranking = []
max_validation_examples = 1000
for i, line in enumerate(tqdm(validation_data)):
    if i == max_validation_examples:
        break
    q, *ex = line
    ranks = rank_candidates(q, ex, embeddings_trained, tokenizer)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

{"model_id":"0fab01afe6f64a9696cf7b2da5143ef4","version_major":2,"vers
ion_minor":0}

```
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_count(wv_ranking, k)))
```


```
{"model_id":"bcacb66d1d9d4ae4876a39ca984ca003","version_major":2,"vers
ion_minor":0}
```


```
DCG@   1: 0.305 | Hits@   1: 0.305
DCG@   5: 0.373 | Hits@   5: 0.434
DCG@  10: 0.395 | Hits@  10: 0.504
DCG@ 100: 0.449 | Hits@ 100: 0.766
DCG@ 500: 0.472 | Hits@ 500: 0.943
DCG@1000: 0.478 | Hits@1000: 1.000
```

С лемматизацией

```
# Создаем общий корпус текстов
corpus = []

for item in tqdm(train_data):
    query, answer = item[0], item[1:]
    # Склейка
    pairs = [tokenizer.tokenize(f'{query} {c}', 're', True) for c in
answer]
    corpus.extend(pairs)
    # break
len(corpus)

from gensim.models import Word2Vec
embeddings_trained = Word2Vec(
    sentences=corpus,        # Корпус токенизированных текстов
    vector_size=200,         # Размерность векторов
    window=3,                # Размер окна контекста
    min_count=3,             # Минимальная частота слов
    workers=4                # Количество потоков
).wv

wv_ranking = []
max_validation_examples = 1000
for i, line in enumerate(tqdm(validation_data)):
    if i == max_validation_examples:
        break
    q, *ex = line
    ranks = rank_candidates(q, ex, embeddings_trained, tokenizer)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)

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_count(wv_ranking, k)))
```

{"model_id":"776905cee6a341b695d24f633f7ec061","version_major":2,"version_minor":0}

{"model_id":"99758bdd36d84c53bc5f9297678e0f1f","version_major":2,"version_minor":0}

{"model_id":"48286774dc2646aca96ed0bfc8c1d696","version_major":2,"version_minor":0}

```
DCG@   1: 0.311 | Hits@   1: 0.311
DCG@   5: 0.380 | Hits@   5: 0.441
DCG@  10: 0.401 | Hits@  10: 0.508
DCG@ 100: 0.457 | Hits@ 100: 0.782
DCG@ 500: 0.478 | Hits@ 500: 0.950
DCG@1000: 0.484 | Hits@1000: 1.000
```

Намеренно плохо обучаем

```
embeddings_trained = Word2Vec(
    sentences=corpus[0],        # Корпус токенизированных текстов
    vector_size=200,         # Размерность векторов
    window=3,                # Размер окна контекста
    min_count=3,             # Минимальная частота слов
    workers=4                # Количество потоков
).wv
wv_ranking = []
max_validation_examples = 1000
for i, line in enumerate(tqdm(validation_data)):
    if i == max_validation_examples:
        break
    q, *ex = line
    ranks = rank_candidates(q, ex, embeddings_trained, tokenizer)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
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_count(wv_ranking, k)))
```

{"model_id":"c0e1ee8d733b4ec6b1dc31b82de27bdb","version_major":2,"version_minor":0}

{"model_id":"0c7476c86b3a4f4e8ad9cdb9c4ab477a","version_major":2,"version_minor":0}

```
DCG@   1: 0.899 | Hits@   1: 0.899
DCG@   5: 0.899 | Hits@   5: 0.899
DCG@  10: 0.899 | Hits@  10: 0.899
DCG@ 100: 0.912 | Hits@ 100: 0.980
DCG@ 500: 0.914 | Hits@ 500: 0.994
DCG@1000: 0.915 | Hits@1000: 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.

Вывод:

- Какой принцип токенизации даёт качество лучше и почему?

Практически оказалось, что токенизация с помощью re дала немного лучшее качество ранжирования. Причина в том, что WordPunctTokenizer создаёт слишком много редких токенов (например, "#", "()", "."), которые либо отсутствуют в словаре Word2Vec, либо вносят шум. Регулярное выражение, наоборот, сохраняет только значимые словоформы, уменьшая разреженность пространства признаков.

- Помогает ли нормализация слов?

Нормализация слов (в частности, приведение к нижнему регистру и лемматизация) положительно влияет на качество решения задачи. Без нормализации одна и та же форма слова может встречаться в корпусе под разными вариантами — например, “running”, “ran”, “runs”. Для модели Word2Vec это разные токены, и их векторные представления обучаются независимо, что приводит к “размыванию” семантики и уменьшению статистической значимости каждого слова. При использовании лемматизации все формы сводятся к базовой — “run”. Это увеличивает частоту встречаемости слов, улучшает устойчивость эмбеддингов и повышает качество при сравнении смыслов предложений с помощью косинусного сходства. В моём эксперименте нормализация дала лучшие значения метрик (Hits@k, DCG@k) по сравнению с вариантом без неё, особенно на малых k. Это объясняется тем, что модель получает более обобщённое и семантически чистое представление текстов, где схожие вопросы действительно ближе в векторном пространстве.

- Какие эмбеддинги лучше справляются с задачей и почему?

Для эмбедингов, обученных на базе train_data, метрики показали результаты хуже, чем на готовых stackoverflow эмбедингах. Это происходит потому что в большинстве случаев вектора нулевые получались, а т.к. в валидационных данных правильный ответ стоит на первом месте в списке вариантов, то в результате сортировки по косинусному расстоянию не происходит и правильный ответ сохраняет первый ранк. То же самое можно наблюдать, если намеренно плохо обучиться на одном предложении, в результате такого обучения у всех похожесть в результате 0 и на валидационной выборке порядок сохраняется, а там на первом месте как раз правильный вариант.

- Почему получилось плохое качество решения задачи?

Простая модель на усреднённых Word2Vec-векторах недостаточно выразительна для задачи семантического ранжирования вопросов. Вопросы на StackOverflow часто содержат ключевые конструкции (“convert string”, “ajax fails”), и усреднение слов теряет их смысловую структуру. В вопросах встречаются специфические токены (C#, jQuery, AJAX, JSON), которых может не быть в обученных эмбеддингах → часть предложений представляется нулевыми векторами.

- Предложите свой подход к решению задачи.

1. Использовать модели типа Sentence-BERT. Эти модели обучаются именно на задачах семантического сходства предложений и учитывают контекст.
2. Добавить нормализацию и предобработку (лемматизация, фильтрация стоп-слов)
3. Комбинировать косинусное сходство с другими метриками (например, BM25).