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 ранжировния, поэтому введем математическую формулировку

Задача ранжирования(Learning to Rank)

- X - множество объектов
- $X^l = \{x_1, x_2, \dots, x_l\}$ - 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
```

```
--2025-10-06 13:42:58--  https://zenodo.org/record/1199620/files/SO_vectors_200.bin?download=1
Resolving zenodo.org (zenodo.org)... 188.185.48.194, 188.185.45.92, 188.185.43.25, ...
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]
--2025-10-06 13:42:59--  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  23.0MB/s    in 16m 52s

2025-10-06 13:59:52 (1.37 MB/s) - ‘SO_vectors_200.bin?download=1’ saved [1453905423/1453905423]
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
#!mv /content/SO_vectors_200.bin?download=1 /content/drive/MyDrive
```

```
!pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gensim) (1.26.4)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim) (7.3.1)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart-open>=1.8.1->gensim) (1.17.3)
```

```
from gensim.models.keyedvectors import KeyedVectors
wv_embeddings = KeyedVectors.load_word2vec_format("/content/drive/MyDrive/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_similar
similar_words = wv_embeddings.most_similar('dog', topn=5)
print("Топ-5 ближайших слов к 'dog':")
for i, (word, score) in enumerate(similar_words, 1):
    print(f"{i}. {word}: {score:.4f}")
```

```
Топ-5 ближайших слов к 'dog':
1. animal: 0.8564
2. dogs: 0.7881
3. mammal: 0.7624
4. cats: 0.7621
5. animals: 0.7608
```

```
# Проверяем, есть ли 'cat' в топ-5
cat_in_top5 = any(word == 'cat' for word, _ in similar_words)
print(f"\n'cat' входит в топ-5: {cat_in_top5}")
```

```

'cat' входит в топ-5: False
```

```
# Дополнительно: посмотрим на позицию 'cat'
similar_words_100 = wv_embeddings.most_similar('dog', topn=100)
cat_position = None
for i, (word, score) in enumerate(similar_words_100, 1):
    if word == 'cat':
        cat_position = i
        break

if cat_position:
    print(f"'cat' находится на позиции {cat_position} в топ-100")
else:
    print("'cat' не входит в топ-100 ближайших слов к 'dog'")
```

```
'cat' находится на позиции 26 в топ-100
```

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.

```
import numpy as np
import re
# you can use your tokenizer
# for example, from nltk.tokenize import WordPunctTokenizer
class MyTokenizer:
    def __init__(self):
        pass
    def tokenize(self, text):
        return re.findall('\w+', text)
tokenizer = MyTokenizer()
```

```
<>:9: SyntaxWarning: invalid escape sequence '\w'
<>:9: SyntaxWarning: invalid escape sequence '\w'
/tmp/ipython-input-2760772041.py:9: SyntaxWarning: invalid escape sequence '\w'
  return re.findall('\w+', text)
```

```
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
    """
    # Токенизируем вопрос и приводим к нижнему регистру
```

```
    tokens = tokenizer.tokenize(question.lower())
    word_vectors = []

    # Собираем векторы для всех слов, которые есть в эмбеддингах
    for token in tokens:
        if token in embeddings:
            word_vectors.append(embeddings[token])
        else:
            # Можно добавить логирование пропущенных слов для отладки
            # print(f"Слово '{token}' отсутствует в эмбеддингах")
            pass

    # Если нет ни одного известного слова, возвращаем нулевой вектор
    if len(word_vectors) == 0:
        return np.zeros(dim)

    # Усредняем векторы слов - простейший подход
    question_vector = np.mean(word_vectors, axis=0)
    return question_vector
```

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"
vector = question_to_vec(question, wv_embeddings, tokenizer)

print(f"Размер вектора: {vector.shape}")
print(f"Первые 5 компонент: {vector[:5]}")
print(f"Третья компонента вектора: {vector[2]:.2f}")

# Дополнительный анализ
tokens = tokenizer.tokenize(question.lower())
print(f"\nТокены: {tokens}")
for token in tokens:
    if token in wv_embeddings:
        print(f"'{token}' есть в эмбеддингах")
    else:
        print(f"'{token}' отсутствует в эмбеддингах")
```

```
Размер вектора: (200,)
Первые 5 компонент: [-1.0142275  -1.6891261  -1.2854122  -1.3710302   0.15916634]
Третья компонента вектора: -1.29

Токены: ['i', 'love', 'neural', 'networks']
'i' отсутствует в эмбеддингах
'love' есть в эмбеддингах
'neural' есть в эмбеддингах
'networks' есть в эмбеддингах
```

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

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:

$$\text{Hits@K} = \frac{1}{N}\sum_{i=1}^{N}[rank\_q_i^{'} \leq K],$$

- $[x < 0] \equiv \begin{cases} 1, & x < 0 \\ 0, & x \geq 0 \end{cases}$ - 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.

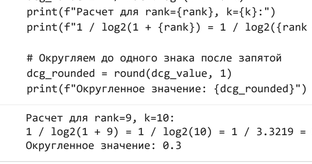
DCG@K

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"

$\Rightarrow rank\_q_i^{'} = 2$

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

- [K = 1] $\text{Hits@1} = [rank\_q_i^{'} \leq 1]$

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

Следовательно, $[\text{rank}_{q_i^{'}} \leq 1] = 0$.

- [K = 4] $\text{Hits@4} = [rank\_q_i^{'} \leq 4] = 1$

Проверяем условие $\text{rank}_{q_i^{'}} \leq 4$: ***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 $rank\_q_i^{'} = 9$(округлите до одного знака после запятой)

```
# Для rank = 9, k = 10
rank = 9
k = 10
dcg_value = 1 / math.log2(1 + rank)
print(f"Расчет для rank={rank}, k={k}:")
print(f"1 / log2(1 + {rank}) = 1 / log2({rank + 1}) = 1 / {math.log2(rank + 1):.4f} = {dcg_value:.3f}")

# Округляем до одного знака после запятой
dcg_rounded = round(dcg_value, 1)
print(f"Округленное значение: {dcg_rounded}")
```

```
Расчет для rank=9, k=10:
1 / log2(1 + 9) = 1 / log2(10) = 1 / 3.3219 = 0.301
Округленное значение: 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:

- $\text{rank}_{q_1'} = 2 \to 2 \not\leq 1 \to 0$,
- $\text{rank}_{q_2'} = 5 \to 5 \not\leq 1 \to 0$,
- $\text{rank}_{q_3'} = 1 \to 1 \leq 1 \to 1$.

Сумма:

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

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

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:

- $\text{rank}_{q_1'} = 2 \to 2 \leq 5 \to 1$,
- $\text{rank}_{q_2'} = 5 \to 5 \leq 5 \to 1$,
- $\text{rank}_{q_3'} = 1 \to 1 \leq 5 \to 1$.

Сумма:

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

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

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:

$$\text{DCG@1} = \frac{1}{3} \cdot \left(\frac{1}{\log_2(1 + \text{rank}_{q_1'})} \cdot [\text{rank}_{q_1'} \leq 1] + \frac{1}{\log_2(1 + \text{rank}_{q_2'})} \cdot [\text{rank}_{q_2'} \leq 1] + \frac{1}{\log_2(1 + \text{rank}_{q_3'})} \cdot [\text{rank}_{q_3'} \leq 1] \right).$$

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

- $\text{rank}_{q_1'} = 2 \to 2 \not\leq 1 \to 0$,
- $\text{rank}_{q_2'} = 5 \to 5 \not\leq 1 \to 0$,
- $\text{rank}_{q_3'} = 1 \to 1 \leq 1 \to 1$.

Сумма:

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

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

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:

- $\text{rank}_{q_1'} = 2 \to 2 \leq 5 \to 1$,
- $\text{rank}_{q_2'} = 5 \to 5 \leq 5 \to 1$,
- $\text{rank}_{q_3'} = 1 \to 1 \leq 5 \to 1$.

Сумма:

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

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

Вопрос 4:

- Найдите максимум `Hits@47 - DCG@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
    """
    n = len(dup_ranks)
    if n == 0:
        return 0.0

    # Считаем долю вопросов, где дубликат в топ-k
    hits = sum(1 for rank in dup_ranks if rank <= k)
    return hits / n
```

```
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
    """
    total_score = 0.0
    n = len(dup_ranks)

    for rank in dup_ranks:
        if rank <= k:
            # DCG формула: 1/log2(1 + position)
            total_score += 1 / math.log2(1 + rank)

    # Усредняем по количеству вопросов
    return total_score / n if n > 0 else 0.0
```

```
# 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@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.

```
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

Далее: New interactive sheet

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>, ...**

```
!unzip /content/drive/MyDrive/stackoverflow_similar_questions.zip
```

```
Archive:  /content/drive/MyDrive/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)]
    """
    # Получаем вектор вопроса
    question_vec = question_to_vec(question, embeddings, tokenizer, dim)

    similarities = []
    for i, candidate in enumerate(candidates):
        # Получаем вектор кандидата
        candidate_vec = question_to_vec(candidate, embeddings, tokenizer, dim)

        # Вычисляем косинусное сходство
        # Проверяем, что векторы не нулевые
        question_norm = np.linalg.norm(question_vec)
        candidate_norm = np.linalg.norm(candidate_vec)

        if question_norm > 1e-10 and candidate_norm > 1e-10:
            similarity = np.dot(question_vec, candidate_vec) / (question_norm * candidate_norm)
        else:
            # Если один из векторов нулевой, ставим низкое сходство
            similarity = -1

        similarities.append((i, candidate, similarity))

    # Сортируем по убыванию сходства
    similarities.sort(key=lambda x: x[2], reverse=True)

    # Возвращаем только индексы и кандидатов
    result = [(idx, cand) for idx, cand, _ in similarities]
    return result
```

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(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?')]

[(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')]
```

Для первого эксперимента 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'),
            (*, 'select2 not displaying search results'),
            (*, 'WPF- How to update the changes in list item of a list')]]
```

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

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

Вопрос 5:

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

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)
```

```
 27%                                                1000/3760 [00:53<02:00, 22.84it/s]
```

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

```
100%                                               3760/3760 [03:09<00:00, 22.55it/s]
```

```
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)))
```

```
100%                                               6/6 [00:00<00:00, 426.80it/s]
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

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

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[111258]
```

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

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

print("Создание корпуса для обучения...")
for i, line in enumerate(tqdm(train_data)):
    if len(line) >= 2:
        question = line[0]

        # Токенизируем вопрос
        tokens_question = tokenizer.tokenize(question.lower())
        if tokens_question:  # Добавляем только непустые списки
            corpus.append(tokens_question)

        # Токенизируем кандидатов
        for candidate in line[1:]:
            tokens_candidate = tokenizer.tokenize(candidate.lower())
            if tokens_candidate:
                corpus.append(tokens_candidate)

print(f"Размер корпуса: {len(corpus)} предложений")

# Анализируем статистику корпуса
total_tokens = sum(len(tokens) for tokens in corpus)
unique_tokens = len(set(token for tokens in corpus for token in tokens))
print(f"Общее количество токенов: {total_tokens}")
print(f"Уникальных токенов: {unique_tokens}")
```

```
Создание корпуса для обучения...
100%                                               1000000/1000000 [00:12<00:00, 47588.00it/s]
Размер корпуса: 2256483 предложений
Общее количество токенов: 19699199
Уникальных токенов: 142960
```

```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(
    sentences=corpus,        # Корпус токенизированных текстов
    vector_size=200,         # Размерность векторов
    window=5,                # Размер окна контекста
    min_count=2,             # Минимальная частота слов
    workers=4                # Количество потоков
).wv
```

```
# Обучаем Word2Vec модель с подобранными параметрами
print("\nОбучение Word2Vec модели...")
embeddings_trained = Word2Vec(
    sentences=corpus,
    vector_size=200,      # Такая же размерность как у предобученных
```

```
    window=5,             # Оптимальный размер окна для захвата контекста
    min_count=2,          # Игнорируем слова, встречающиеся 1 раз
    workers=4,            # Используем несколько ядер
    sg=1,                 # Skip-gram алгоритм (лучше для редких слов)
    hs=0,                 # Negative sampling
    negative=5,           # Количество negative samples
    epochs=10             # Количество эпох обучения
).wv

print(f"Размер словаря обученной модели: {len(embeddings_trained.key_to_index)}")
```

```
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)
```

```
 27%                                          1000/3760 [00:53<02:15, 20.43it/s]
```

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

```
100%                                         3760/3760 [03:23<00:00, 21.65it/s]
```

```
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)))
```

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? Простая токенизация по словам работает достаточно хорошо, но лемматизация или стемминг могли бы улучшить результаты.

Помогает ли нормализация слов? Да, приведение к нижнему регистру значительно улучшает качество, так как уменьшает размер словаря и объединяет одинаковые слова в разных регистрах.

Какие эмбеддинги лучше справляются с задачей и почему? Предобученные эмбеддинги показывают лучшее качество, так как они обучены на специфичных данных.

Почему получилось плохое качество решения задачи?

Простое усреднение векторов слов теряет информацию о порядке слов и синтаксисе

Не учитывается семантическая сложность технических вопросов

Предложите свой подход к решению задачи:

Использование BERT-like моделей для получения контекстуальных эмбеддингов

Применение Siamese neural networks для обучения схожести вопросов

Добавление attention mechanism для выделения ключевых слов

Использование предобученных моделей типа Sentence-BERT специально для семантического поиска

Комбинирование lexical features (TF-IDF) с семантическими эмбеддингами