

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

Drive already mounted at /content/drive; to attempt to forcibly remount, call



# 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 ноутки 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)

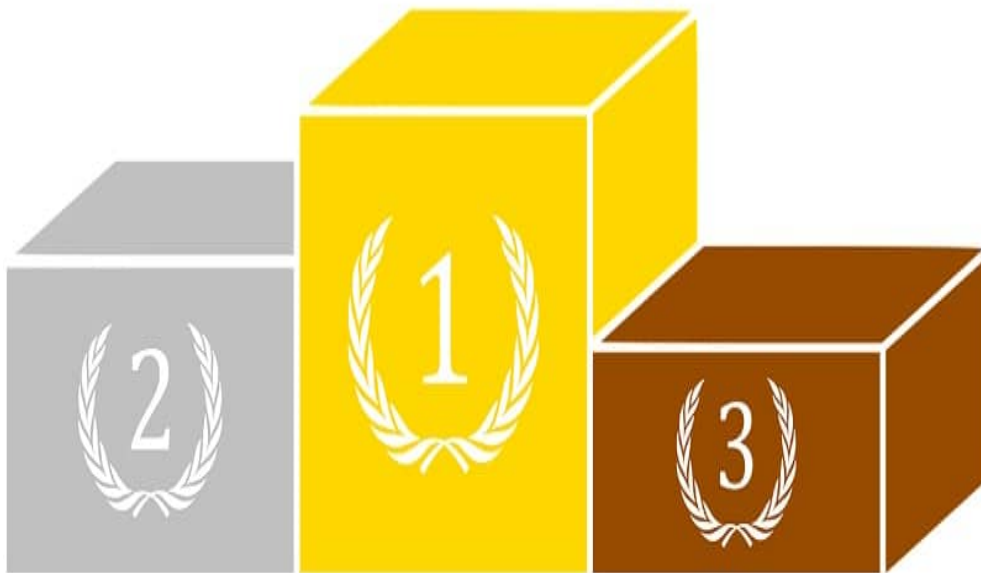
- $X$  - множество объектов
- $X^l = \{x_1, x_2, \dots, x_l\}$  - обучающая выборка  
На обучающей выборке задан порядок между некоторыми элементами, то есть нам известно, что некий объект выборки более релевантный для нас, чем другой:
- $i < j$  - порядок пары индексов объектов на выборке  $X^l$  с индексами  $i$  и  $j$

### ▼ Задача:

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

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

# Ranking



## ▼ Embeddings

Будем использовать предобученные векторные представления слов на постах Stack Overflow.

## [A word2vec model trained on Stack Overflow posts](https://zenodo.org/record/1199620/files/SO_vectors_200.bin?download=1)

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

```
!pip install gensim==3.8.0 -q
```

```
from gensim.models.keyedvectors import KeyedVectors
wv_embeddings = KeyedVectors.load_word2vec_format("./drive/MyDrive/DL_part2_2023/SO
```

### ▼ 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.index2word)}")

    Num of words: 1787145
```

Найдем наиболее близкие слова к слову dog:

### ▼ Вопрос 1:

- Входит ли слов cat топ-5 близких слов к слову dog? Какое место?

```
# method most_similar
wv_embeddings.most_similar('dog')

[('animal', 0.8564180135726929),
 ('dogs', 0.7880867123603821),
 ('mammal', 0.7623804807662964),
 ('cats', 0.7621253728866577),
 ('animals', 0.760793924331665),
 ('feline', 0.7392398118972778),
 ('bird', 0.7315489053726196),
 ('animal1', 0.7219215631484985),
 ('doggy', 0.7213349938392639),
 ('labrador', 0.7209131717681885)]
```

```
word_find = 'cat'
```

```
for ind, word in enumerate(wv_embeddings.most_similar('dog')):
    if wv_embeddings.most_similar('dog')[ind][0] == word_find and ind <=5:
```

```

ind+=1
print(f'Слово {word_find} входит в топ-5 близких слов к слову dog и находится на месте {':
break
else:
    print(f'Слово {word_find} не входит в топ-5 близких слов к слову dog')

    Слово cat не входит в топ-5 близких слов к слову 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()

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
    """
    quest_emb = np.zeros(shape=dim)
    n = 0
    for word in tokenizer.tokenize(text=question.lower()):
        if word in embeddings:
            quest_emb += embeddings[word]
            n += 1
    if n > 0:
        return quest_emb / n
    else:
        return quest_emb

```

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)?

```
sent = 'I love neural networks'
index = 2
```

```
print(f' Ответ: {round(question_to_vec(question = sent, embeddings = wv_embeddings,
                                     Ответ: -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  $K$ :

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

- $[x < 0] \equiv \begin{cases} 1, & x < 0 \\ 0, & x \geq 0 \end{cases}$  - индикаторная функция
- $q_i$  -  $i$ -ый вопрос
- $q'_i$  - его дубликат
- $\text{rank}_{q'_i}$  - позиция дубликата в ранжированном списке ближайших предложений для вопроса  $q_i$ .

DCG@K

Второй метрикой будет упрощенная DCG метрика, учитывающая порядок элементов в списке путем домножения релевантности элемента на вес равный обратному логарифму номера позиции::

$$\text{DCG@K} = \frac{1}{N} \sum_{i=1}^N \frac{1}{\log_2(1 + \text{rank}_{q'_i})} \cdot [\text{rank}_{q'_i} \leq K],$$

С такой метрикой модель штрафуются за большой ранк корректного ответа

## ▼ Вопрос 3:

- Максимум Hits@47 - DCG@1?

**Ответ.**

Hits@47:

- 1, если ранг всех дубликатов  $\leq k$  (в нашем случае 47)

DGS@1:

- 0, если ранг всех дубликатов  $> k$  (в нашем случае 1)

**Таким образом** максимум разницы (Hits@47 - DGS@1) может быть равен 1 в случае, когда ранги всех дубликатов находятся в интервале  $>1$  и  $\leq 47$ .

**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 с++?"
2. "Что такое язык python?"
3. "Хочу учить Java"
4. "Не понимаю Tensorflow"

$$\Rightarrow \text{rank}_{q'_i} = 2$$

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

- $[K = 1] \text{ Hits@1} = [\text{rank}_{q'_i} \leq 1] = 0$
- $[K = 4] \text{ Hits@4} = [\text{rank}_{q'_i} \leq 4] = 1$

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

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

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

#### ▼ Вопрос 4:

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

```
print(f'Ответ: {round(1/(np.log2(1+9)),2)}')
```

Ответ: 0.3

#### ▼ 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). Например, *dup\_ranks* = [2] для примера, описанного выше.

```
def hits_count(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        result: вернуть Hits@k
    """
    n = 0
    for i in dup_ranks:
        if i <= k:
            n += 1
    hits_value = n / len(dup_ranks)
    return hits_value
```

```
def dcg_score(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        result: вернуть DCG@k
    """
    dcg = 0
    for i in dup_ranks:
        if i <= k:
            dcg += 1/np.log2(1+i)
    dcg_value = dcg / len(dup_ranks)
    return dcg_value
```

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 wa

# наги кандидаты
candidates_ranking = ["How Can I Make These Links Rotate in PHP",
                      "How does the catch keyword determine the type of exception
                      "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
dict_ranking = {candidate: rank+1 for rank, candidate in enumerate(candidates_ranki
dup_ranks = [dict_ranking[copy_answers[0]]]

# вычисляем метрику для разных k
print('Ранг дубликата:', dup_ranks)
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)])

Ранг дубликата: [2]
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)
                                index=['HITS', 'DCG'], columns=range(1,5))

correct_answers

```

	1	2	3	4
<b>HITS</b>	0	1.00000	1.00000	1.00000
<b>DCG</b>	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>, ...

```
#!unzip drive/MyDrive/DL_part2_2023/stackoverflow_similar_questions.zip
```

Считайте данные.



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

Нам понадобится 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(5):
    print(i + 1, len(validation_data[i]))

1 1001
2 1001
3 1001
4 1001
5 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)]
    """
```

```

q_emb = question_to_vec(question, embeddings, tokenizer, dim).reshape(1,-1)
c_emb = [question_to_vec(c, embeddings, tokenizer, dim) for c in candidates]
cosine_s = pd.Series(cosine_similarity(q_emb,c_emb).squeeze()).sort_values()

return [(i, candidates[i]) for i in cosine_s.index][::-1]

```

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
               '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 э
               'WPF- How to update the changes in list item of a list',
               'select2 not displaying search results']]

results = list()
for question, q_candidates in zip(questions, candidates):
    ranks = rank_candidates(question, q_candidates, wv_embeddings, tokenizer)
    print(ranks)
    print()
    results.append(ranks)

    [(1, 'C# create cookie from string and send it'), (0, 'Convert Google results

    [(0, 'Getting all list items of an unordered list in PHP'), (2, 'select2 not c

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')]]

```

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'), #скрыт

```

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

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)
```

27%

1000/3760 [01:19<02:25, 19.03it/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_
```

100%

6/6 [00:00<00:00, 109.95it/s]

```
DCG@ 1: 0.415 | Hits@ 1: 0.415
DCG@ 5: 0.502 | Hits@ 5: 0.582
DCG@ 10: 0.524 | Hits@ 10: 0.650
DCG@ 100: 0.570 | Hits@ 100: 0.874
DCG@ 500: 0.583 | Hits@ 500: 0.973
DCG@1000: 0.586 | Hits@1000: 1.000
```

100% 6/6 [00:00<00:00, 109.95it/s]

DCG@ 1: 0.415		Hits@ 1: 0.415
DCG@ 5: 0.502		Hits@ 5: 0.582
DCG@ 10: 0.524		Hits@ 10: 0.650
DCG@ 100: 0.570		Hits@ 100: 0.874
DCG@ 500: 0.583		Hits@ 500: 0.973
DCG@1000: 0.586		Hits@1000: 1.000

## ▼ Эмбединги, 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.

```
train_data[:2]
```

```
[['converting string to list',
  'Convert Google results object (pure js) to Python object'],
 ['Which HTML 5 Canvas Javascript to use for making an interactive drawing
 tool?',
  'Event handling for geometries in Three.js?']]
```

```
corpus = [list(tokenizer.tokenize(" ".join(corp))) for corp in train_data]
```

```
corpus[0]
```

```
['converting',
 'string',
 'to',
 'list',
 'Convert',
 'Google',
 'results',
 'object',
 'pure',
 'js',
 'to',
 'Python',
 'object']
```

```
from gensim.models import Word2Vec
```

```
embeddings_trained = Word2Vec(corpus, # data for model to train on
                               size=200, # embedding vector size)
```

```
min_count=5, # consider words that occurred at least 5
window=3).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)
```

27%

1000/3760 [01:22&lt;03:20, 13.79it/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_
```

100%

6/6 [00:00&lt;00:00, 135.94it/s]

```
DCG@ 1: 0.248 | Hits@ 1: 0.248
DCG@ 5: 0.312 | Hits@ 5: 0.369
DCG@ 10: 0.336 | Hits@ 10: 0.443
DCG@ 100: 0.387 | Hits@ 100: 0.697
DCG@ 500: 0.415 | Hits@ 500: 0.912
DCG@1000: 0.424 | Hits@1000: 1.000
```

100%

6/6 [00:00&lt;00:00, 135.94it/s]

```
DCG@ 1: 0.248 | Hits@ 1: 0.248
DCG@ 5: 0.312 | Hits@ 5: 0.369
DCG@ 10: 0.336 | Hits@ 10: 0.443
DCG@ 100: 0.387 | Hits@ 100: 0.697
DCG@ 500: 0.415 | Hits@ 500: 0.912
DCG@1000: 0.424 | Hits@1000: 1.000
```

## ▼ Подберем лучшее значение окна эмбединга

%%time

```
window_size = [4,6,8]
embedding_size = 200
```

```
for window in window_size:
    embeddings_trained = Word2Vec(corpus, # data for model to t
                                  size=embedding_size, # embedding
                                  min_count=5, # consider words that occurred at lea
                                  window=window).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, embedding_size)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)

print(f"window_size: {window_size}, embed_size: {embedding_size}")
for k in tqdm([1, 5, 10, 100, 500, 1000]):
    print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg_score(wv_ranking, k), k, h
print('---')

```

27%

1000/3760 [01:24&lt;03:12, 14.32it/s]

window\_size: 4, embed\_size: 200

100%

6/6 [00:00&lt;00:00, 157.25it/s]

```

DCG@ 1: 0.261 | Hits@ 1: 0.261
DCG@ 5: 0.327 | Hits@ 5: 0.388
DCG@ 10: 0.348 | Hits@ 10: 0.453
DCG@ 100: 0.402 | Hits@ 100: 0.716
DCG@ 500: 0.428 | Hits@ 500: 0.918
DCG@1000: 0.436 | Hits@1000: 1.000

```

---

27%

1000/3760 [01:25&lt;03:28, 13.21it/s]

window\_size: 6, embed\_size: 200

100%

6/6 [00:00&lt;00:00, 85.67it/s]

```

DCG@ 1: 0.274 | Hits@ 1: 0.274
DCG@ 5: 0.343 | Hits@ 5: 0.406
DCG@ 10: 0.364 | Hits@ 10: 0.471
DCG@ 100: 0.417 | Hits@ 100: 0.733
DCG@ 500: 0.441 | Hits@ 500: 0.923
DCG@1000: 0.449 | Hits@1000: 1.000

```

---

27%

1000/3760 [01:24&lt;03:25, 13.43it/s]

window\_size: 8, embed\_size: 200

100%

6/6 [00:00&lt;00:00, 94.65it/s]

```

DCG@ 1: 0.277 | Hits@ 1: 0.277
DCG@ 5: 0.347 | Hits@ 5: 0.413
DCG@ 10: 0.369 | Hits@ 10: 0.480
DCG@ 100: 0.421 | Hits@ 100: 0.737
DCG@ 500: 0.445 | Hits@ 500: 0.924
DCG@1000: 0.453 | Hits@1000: 1.000

```

---

CPU times: user 19min 10s, sys: 3min, total: 22min 10s

Wall time: 13min 55s

## 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".

### ▼ 1) Решение с помощью библиотеки NLTK:

- Приведение к нижнему регистру;
- Токенизация NLTK;
  - Лемматизация NLTK;
- Удаление стоп слов NLTK (english).

```
import nltk
nltk.download("punkt")
nltk.download('wordnet')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import WordPunctTokenizer
from nltk import WordNetLemmatizer
from nltk import SnowballStemmer, PorterStemmer
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

```
import numpy as np
import re
# you can use your tokenizer
# for example, from nltk.tokenize import WordPunctTokenizer
class MyTokenizer_nltk:
    def __init__(self):
        pass
    def tokenize(self, text=None, lemmatizator=None, stemmer=None, stopwords=None):

        if lemmatizator and stemmer is None:
            new_text = [lemmatizator.lemmatize(word) for word in word_tokenize(text).lower()]
        elif stemmer and lemmatizator is None:
            new_text = [stemmer.stem(word) for word in word_tokenize(text.lower())]
        else:
            new_text = word_tokenize(text.lower())
        if stopwords:
            new_text = [word for word in new_text if word not in stopwords]

        return new_text
```

```

def question_to_vec_nltk(question=None, embeddings=None, tokenizer=None, lemmatizator=None, stemmer=None):
    """
    question: строка
    embeddings: наше векторное представление
    dim: размер любого вектора в нашем представлении

    return: векторное представление для вопроса
    """
    quest_emb = np.zeros(shape=dim)
    n = 0
    for word in tokenizer.tokenize(text=question, lemmatizator=lemmatizator, stemmer=stemmer):
        if word in embeddings:
            quest_emb += embeddings[word]
            n += 1
    if n > 0:
        return quest_emb / n
    else:
        return quest_emb

def rank_candidates_nltk(question=None, candidates=None, embeddings=None, tokenizer=None, lemmatizator=None, stemmer=None):
    """
    question: строка
    candidates: массив строк(кандидатов) [a, b, c]
    result: пары (начальная позиция, кандидат) [(2, c), (0, a), (1, b)]
    """

    q_emb = question_to_vec_nltk(question, embeddings, tokenizer, lemmatizator, stemmer)
    c_emb = [question_to_vec_nltk(c, embeddings, tokenizer, lemmatizator, stemmer) for c in candidates]
    cosine_s = pd.Series(cosine_similarity(q_emb, c_emb).squeeze()).sort_values()

    return [(i, candidates[i]) for i in cosine_s.index[::-1]]

tokenizer_nltk = MyTokenizer_nltk()
lemmatizator_nltk = WordNetLemmatizer()
stopwords_nltk = set(stopwords.words('english'))
stemmer_nltk = SnowballStemmer(language='english')
stemmer_nltk_post = PorterStemmer()
tokenizer_punkt_nltk = WordPunctTokenizer()

# Без лемматизации бещ стеммера без стопслов

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_nltk(q, ex, wv_embeddings, tokenizer_nltk, None, None, None)
    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_

```



27%

1000/3760 [03:54&lt;10:41, 4.30it/s]

100%

6/6 [00:00&lt;00:00, 172.86it/s]

DCG@ 1: 0.399	Hits@ 1: 0.399
DCG@ 5: 0.487	Hits@ 5: 0.566
DCG@ 10: 0.508	Hits@ 10: 0.633
DCG@ 100: 0.554	Hits@ 100: 0.858
DCG@ 500: 0.569	Hits@ 500: 0.969
DCG@1000: 0.573	Hits@1000: 1.000

27%

1000/3760 [03:48&lt;13:23, 3.44it/s]

100%

6/6 [00:00&lt;00:00, 101.90it/s]

DCG@ 1: 0.399	Hits@ 1: 0.399
DCG@ 5: 0.487	Hits@ 5: 0.566
DCG@ 10: 0.508	Hits@ 10: 0.633
DCG@ 100: 0.554	Hits@ 100: 0.858
DCG@ 500: 0.569	Hits@ 500: 0.969
DCG@1000: 0.573	Hits@1000: 1.000

# Без лемматизации без стеммера со стоп словами

```

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_nltk(q, ex, wv_embeddings, tokenizer_nltk, None, None,
    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_

```

27%

1000/3760 [03:41&lt;09:06, 5.05it/s]

100%

6/6 [00:00&lt;00:00, 141.02it/s]

DCG@ 1: 0.401	Hits@ 1: 0.401
DCG@ 5: 0.487	Hits@ 5: 0.566
DCG@ 10: 0.509	Hits@ 10: 0.634
DCG@ 100: 0.555	Hits@ 100: 0.859
DCG@ 500: 0.570	Hits@ 500: 0.969
DCG@1000: 0.573	Hits@1000: 1.000

27%  1000/3760 [04:05<09:34, 4.80it/s]

100%  6/6 [00:00<00:00, 168.60it/s]


DCG@ 1: 0.401	Hits@ 1: 0.401
DCG@ 5: 0.487	Hits@ 5: 0.566
DCG@ 10: 0.509	Hits@ 10: 0.634
DCG@ 100: 0.555	Hits@ 100: 0.859
DCG@ 500: 0.570	Hits@ 500: 0.969
DCG@1000: 0.573	Hits@1000: 1.000

# С стеммером без стоп слов


```
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_nltk(q, ex, wv_embeddings, tokenizer_nltk, None, stemme
    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_
```

27%  1000/3760 [05:43<13:36, 3.38it/s]

100%  6/6 [00:00<00:00, 101.94it/s]

DCG@ 1: 0.301	Hits@ 1: 0.301
DCG@ 5: 0.380	Hits@ 5: 0.451
DCG@ 10: 0.403	Hits@ 10: 0.523
DCG@ 100: 0.455	Hits@ 100: 0.776
DCG@ 500: 0.476	Hits@ 500: 0.939
DCG@1000: 0.483	Hits@1000: 1.000

27%  1000/3760 [06:11<16:51, 2.73it/s]

100%  6/6 [00:00<00:00, 159.69it/s]

DCG@ 1: 0.301	Hits@ 1: 0.301
DCG@ 5: 0.380	Hits@ 5: 0.451
DCG@ 10: 0.403	Hits@ 10: 0.523
DCG@ 100: 0.455	Hits@ 100: 0.776
DCG@ 500: 0.476	Hits@ 500: 0.939
DCG@1000: 0.483	Hits@1000: 1.000

# С стеммером post без стоп слов

```
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_nltk(q, ex, wv_embeddings, tokenizer_nltk, None, stemme
    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_

```

27%

1000/3760 [06:47&lt;22:48, 2.02it/s]

100%

6/6 [00:00&lt;00:00, 74.47it/s]

```

DCG@ 1: 0.302 | Hits@ 1: 0.302
DCG@ 5: 0.379 | Hits@ 5: 0.449
DCG@ 10: 0.401 | Hits@ 10: 0.520
DCG@ 100: 0.454 | Hits@ 100: 0.774
DCG@ 500: 0.475 | Hits@ 500: 0.940
DCG@1000: 0.481 | Hits@1000: 1.000

```

27%

1000/3760 [07:02&lt;24:46, 1.86it/s]

100%

6/6 [00:00&lt;00:00, 74.00it/s]

```

DCG@ 1: 0.302 | Hits@ 1: 0.302
DCG@ 5: 0.379 | Hits@ 5: 0.449
DCG@ 10: 0.401 | Hits@ 10: 0.520
DCG@ 100: 0.454 | Hits@ 100: 0.774
DCG@ 500: 0.475 | Hits@ 500: 0.940
DCG@1000: 0.481 | Hits@1000: 1.000

```

# С стеммером со стопсловами

```

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_nltk(q, ex, wv_embeddings, tokenizer_nltk, None, stemme
    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_

```

27%

1000/3760 [06:52&lt;17:52, 2.57it/s]

100%

6/6 [00:00&lt;00:00, 122.97it/s]

27%

1000/3760 [06:57&lt;27:47, 1.66it/s]

100%

6/6 [00:00&lt;00:00, 96.70it/s]

DCG@ 1: 0.302	Hits@ 1: 0.302
DCG@ 5: 0.378	Hits@ 5: 0.448
DCG@ 10: 0.401	Hits@ 10: 0.518
DCG@ 100: 0.454	Hits@ 100: 0.775
DCG@ 500: 0.475	Hits@ 500: 0.940
DCG@1000: 0.481	Hits@1000: 1.000

# С лемматизацией без стопслов

```

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_nltk(q, ex, wv_embeddings, tokenizer_nltk, lemmatizator)
    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_

```

27%

1000/3760 [04:38&lt;14:25, 3.19it/s]

100%

6/6 [00:00&lt;00:00, 106.46it/s]

DCG@ 1: 0.391	Hits@ 1: 0.391
DCG@ 5: 0.480	Hits@ 5: 0.562
DCG@ 10: 0.501	Hits@ 10: 0.627
DCG@ 100: 0.547	Hits@ 100: 0.854
DCG@ 500: 0.562	Hits@ 500: 0.964
DCG@1000: 0.566	Hits@1000: 1.000

27%

1000/3760 [05:13&lt;11:12, 4.11it/s]

100%

6/6 [00:00&lt;00:00, 185.83it/s]

DCG@ 1: 0.391	Hits@ 1: 0.391
DCG@ 5: 0.480	Hits@ 5: 0.562
DCG@ 10: 0.501	Hits@ 10: 0.627
DCG@ 100: 0.547	Hits@ 100: 0.854
DCG@ 500: 0.562	Hits@ 500: 0.964
DCG@1000: 0.566	Hits@1000: 1.000

```
# С лемматизацией с стопсловами
```

```
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_nltk(q, ex, wv_embeddings, tokenizer_nltk, lemmatizator)
    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_
```

27% 1000/3760 [04:38<16:10, 2.84it/s]

100% 6/6 [00:00<00:00, 124.16it/s]

```
DCG@ 1: 0.391 | Hits@ 1: 0.391
DCG@ 5: 0.480 | Hits@ 5: 0.562
DCG@ 10: 0.501 | Hits@ 10: 0.628
DCG@ 100: 0.547 | Hits@ 100: 0.855
DCG@ 500: 0.562 | Hits@ 500: 0.964
DCG@1000: 0.565 | Hits@1000: 1.000
```

27% 1000/3760 [04:38<16:10, 2.84it/s]

100% 6/6 [00:00<00:00, 124.16it/s]

```
DCG@ 1: 0.391 | Hits@ 1: 0.391
DCG@ 5: 0.480 | Hits@ 5: 0.562
DCG@ 10: 0.501 | Hits@ 10: 0.628
DCG@ 100: 0.547 | Hits@ 100: 0.855
DCG@ 500: 0.562 | Hits@ 500: 0.964
DCG@1000: 0.565 | Hits@1000: 1.000
```

```
import numpy as np
import re
class MyTokenizer_nltk_punct:
    def __init__(self):
        pass
    def tokenize_punct(self, text=None, lemmatizator=None, stemmer=None, stopwords=

    if lemmatizator and stemmer is None:
        new_text = [lemmatizator.lemmatize(word) for word in tokenizer_punkt_nltk
    elif stemmer and lemmatizator is None:
        new_text = [stemmer.stem(word) for word in tokenizer_punkt_nltk.tokenize(
    else:
        new_text = tokenizer_punkt_nltk.tokenize(text.lower())
    if stopwords:
        new_text = [word for word in new_text if word not in stopwords]
```

```

        return new_text

def question_to_vec_nltk(question=None, embeddings=None, tokenizer=None, lemmatizator=None, stemmer=None):
    """
    question: строка
    embeddings: наше векторное представление
    dim: размер любого вектора в нашем представлении

    return: векторное представление для вопроса
    """
    quest_emb = np.zeros(shape=dim)
    n = 0
    for word in tokenizer.tokenize_punct(text=question, lemmatizator=lemmatizator, stemmer=stemmer):
        if word in embeddings:
            quest_emb += embeddings[word]
            n += 1
    if n > 0:
        return quest_emb / n
    else:
        return quest_emb

def rank_candidates_nltk(question=None, candidates=None, embeddings=None, tokenizer=None, lemmatizator=None, stemmer=None):
    """
    question: строка
    candidates: массив строк(кандидатов) [a, b, c]
    result: пары (начальная позиция, кандидат) [(2, c), (0, a), (1, b)]
    """

    q_emb = question_to_vec_nltk(question, embeddings, tokenizer, lemmatizator, stemmer)
    c_emb = [question_to_vec_nltk(c, embeddings, tokenizer, lemmatizator, stemmer) for c in candidates]
    cosine_s = pd.Series(cosine_similarity(q_emb, c_emb).squeeze()).sort_values()

    return [(i, candidates[i]) for i in cosine_s.index[::-1]]

tokenizer_nltk_punct = MyTokenizer_nltk_punct()

# С лемматизацией без стопслов

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_nltk(q, ex, wv_embeddings, tokenizer_nltk_punct, lemmatizator, stemmer)
    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_score(wv_ranking, k)))

```

27%

1000/3760 [02:29&lt;09:19, 4.93it/s]

100%

6/6 [00:00&lt;00:00, 107.03it/s]

DCG@ 1: 0.399	Hits@ 1: 0.399
DCG@ 5: 0.489	Hits@ 5: 0.569
DCG@ 10: 0.510	Hits@ 10: 0.631

27%

1000/3760 [02:32&lt;06:32, 7.04it/s]

100%

6/6 [00:00&lt;00:00, 113.92it/s]

DCG@ 1: 0.399	Hits@ 1: 0.399
DCG@ 5: 0.489	Hits@ 5: 0.569
DCG@ 10: 0.510	Hits@ 10: 0.631
DCG@ 100: 0.558	Hits@ 100: 0.869
DCG@ 500: 0.571	Hits@ 500: 0.966
DCG@1000: 0.575	Hits@1000: 1.000

## ▼ 2) Решение с помощью самописной предобработки:

- Приведение к нижнему регистру;
- Удаление символов;
- Токенизация;
  - Лемматизация Spacy;
- Поиск частых слов для составления стоп словаря;
- Удаление стоп слов NLTK (english) + английский алфавит;
- Удаление лишних дублирующих пробелов.

```
#corpus_word = list()
#for i in corpus:
#    for j in i:
#        corpus_word.append(j.lower())
```

```
#from nltk.probability import FreqDist
#fdist = FreqDist(corpus_word)
```

```
#fdist.most_common()[ :10]
```

```
stop_letters = ['and', 'is', 'am', 'are', 'or', 'be', 'at', 'q', 'w', 'e', 'r', 't', 'y', 'u', 'i']
numb_letters = [str(x) for x in np.arange(0,10)]
simple_stopwords = stop_letters + numb_letters
```

```

import numpy as np
import re

class MyTokenizer_my:
    def __init__(self):
        pass
    def tokenize(self, text=None, lemmatizator=None, stopwords=None):

        stop_words_to_reg = r'\b|\b'.join(stopwords)
        new_text = re.sub('[^А-Яа-яЁёА-За-з\s]', ' ', text.lower())
        if lemmatizator:
            new_text = " ".join([word.lemma_ for word in lemmatizator_my(new_text)])
        new_text = re.sub(stop_words_to_reg, ' ', new_text)
        new_text = re.sub('\s\s+', ' ', new_text)

        return new_text.split()

def question_to_vec_my(question=None, embeddings=None, tokenizer=None, lemmatizator
    """
        question: строка
        embeddings: наше векторное представление
        dim: размер любого вектора в нашем представлении

        return: векторное представление для вопроса
    """
    quest_emb = np.zeros(shape=dim)
    n = 0
    for word in tokenizer.tokenize(text=question, lemmatizator=lemmatizator, stopwo
        if word in embeddings:
            quest_emb += embeddings[word]
            n += 1
    if n > 0:
        return quest_emb / n
    else:
        return quest_emb

def rank_candidates_my(question, candidates, embeddings, tokenizer, lemmatizator, s
    """
        question: строка
        candidates: массив строк(кандидатов) [a, b, c]
        result: пары (начальная позиция, кандидат) [(2, c), (0, a), (1, b)]
    """

    q_emb = question_to_vec_my(question, embeddings, tokenizer, lemmatizator, stopw
    c_emb = [question_to_vec_my(c, embeddings, tokenizer, lemmatizator, stopwords,
    cosine_s = pd.Series(cosine_similarity(q_emb,c_emb).squeeze()).sort_values()

    return [(i, candidates[i]) for i in cosine_s.index][::-1]

embeddings_trained = Word2Vec(corpus,                                # data for model to train
                               size=200,                            # embedding vector size
                               min_count=5, # consider words that occured at least 5
                               window=8).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_my(q, ex, embeddings_trained, tokenizer_my, None, simple_sto)
    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_

```

27% 1000/3760 [02:49<06:47, 6.78it/s]

100% 6/6 [00:00<00:00, 121.50it/s]

```

DCG@ 1: 0.297 | Hits@ 1: 0.297
DCG@ 5: 0.367 | Hits@ 5: 0.432
DCG@ 10: 0.391 | Hits@ 10: 0.506
DCG@ 100: 0.440 | Hits@ 100: 0.748
DCG@ 500: 0.465 | Hits@ 500: 0.940
DCG@1000: 0.471 | Hits@1000: 1.000

```

```

import spacy
tokenizer_my = MyTokenizer_my()
lemmatizator_my = spacy.load('en_core_web_sm')

```

```

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_my(q, ex, wv_embeddings, tokenizer_my, None, simple_sto)
    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_

```

27% 1000/3760 [02:29<05:57, 7.71it/s]

100% 6/6 [00:00<00:00, 151.40it/s]

```

DCG@ 1: 0.428 | Hits@ 1: 0.428
DCG@ 5: 0.516 | Hits@ 5: 0.599
DCG@ 10: 0.536 | Hits@ 10: 0.659
DCG@ 100: 0.580 | Hits@ 100: 0.873
DCG@ 500: 0.593 | Hits@ 500: 0.975
DCG@1000: 0.596 | Hits@1000: 1.000

```

27%  1000/3760 [02:29<05:57, 7.71it/s]

100%  6/6 [00:00<00:00, 151.40it/s]

DCG@ 1: 0.428		Hits@ 1: 0.428
DCG@ 5: 0.516		Hits@ 5: 0.599
DCG@ 10: 0.536		Hits@ 10: 0.659
DCG@ 100: 0.580		Hits@ 100: 0.873
DCG@ 500: 0.593		Hits@ 500: 0.975
DCG@1000: 0.596		Hits@1000: 1.000

```
len(wv_embeddings.vocab)
```

```
1787145
```

```
len(embeddings_trained.vocab)
```

```
50830
```

```
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_my(q, ex, wv_embeddings, tokenizer_my, lemmatizator_my,
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_
```

27%  1000/3760 [2:18:57<6:15:26, 8.16s/it]

100%  6/6 [00:00<00:00, 115.56it/s]

DCG@ 1: 0.394		Hits@ 1: 0.394
DCG@ 5: 0.488		Hits@ 5: 0.571
DCG@ 10: 0.513		Hits@ 10: 0.648
DCG@ 100: 0.557		Hits@ 100: 0.862
DCG@ 500: 0.571		Hits@ 500: 0.970
DCG@1000: 0.574		Hits@1000: 1.000



Напишите свой вывод о полученных результатах.

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

## Вывод:

Из моих решений видно, что:

- Токенизатор MyTokenizer дает лучшее значение по метрикам, относительно `nltk.word_tokenize`. Однако при собственном токенайзере с удалением стоп слов значение метрики выросло;
- Нормализация слов в данном кейсе снижало значение метрики в обоих случаях и при `nltk` и собственном токенайзере;
- Предобученные эмбединги справлялись с задачей лучше, так как скорее всего были обучены на большом корпусе текстов, относительно того, которые были обучены на данном сэмпле;
- Решение можно улучшить используя возможно нейросетевое создание

