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.

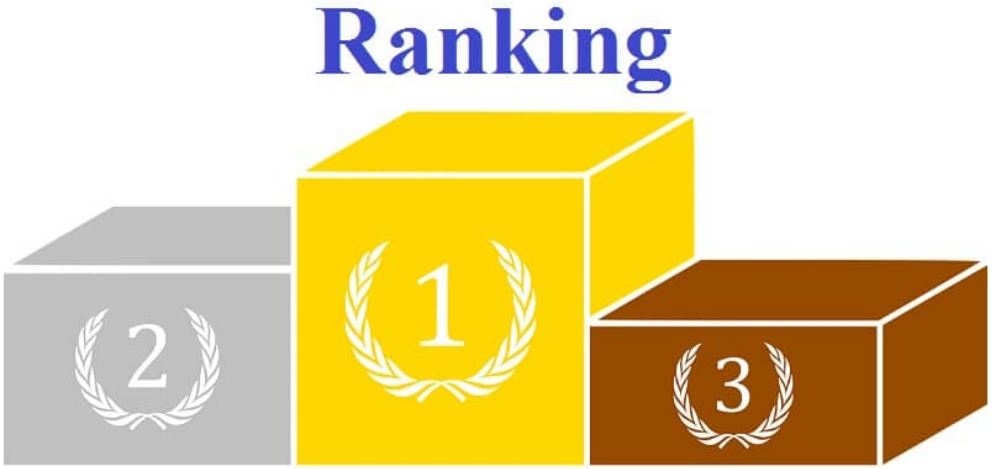
Задача поиска схожих по смыслу предложений

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**
- **$X^l = \{x_1, x_2, ..., x_l\}$ - 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 : $X \rightarrow R$ 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

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

```
--2021-02-19 11:05:50--  https://zenodo.org/record/1199620/files/SO_vectors_200.bin?download=1
Resolving zenodo.org (zenodo.org)... 137.138.76.77
Connecting to zenodo.org (zenodo.org)|137.138.76.77|:443... connected.
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  13.5MB/s    in 3m 23s
```

```
2021-02-19 11:09:15 (6.82 MB/s) - ‘SO_vectors_200.bin?download=1’ saved [1453905423/1453905423]
```

In [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

In [3]:
```
word = 'dog'
if word in wv_embeddings:
    print(wv_embeddings[word].dtype, wv_embeddings[word].shape)
```

```
float32 (200,)
```

In [4]:
```
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?**

In [5]:
```
# method most_simmilar
'''your code'''
wv_embeddings.most_similar("dog")
```

Out[5]:
```
[('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)]
```

Ответ 1:

а) входит

б) место - 4 (с индексом 3, если считать от 0)

Векторные представления текста

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

In [2]:
```
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()
```

In [7]:
```
# example
tokenizer.tokenize("dim: размер любого вектора в нашем представлении")
```

Out[7]:
```
['dim', 'размер', 'любого', 'вектора', 'в', 'нашем', 'представлении']
```

In [3]:
```
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
    """
    embedding_dim = embeddings.vectors.shape[1]
    features = np.zeros([embedding_dim], dtype='float32')

    n = 0
    for word in tokenizer.tokenize(question):
        if word in embeddings:
            features += embeddings[f'{word}']
            n += 1
    if n != 0:
        features /= n
    return features
```

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

Вопрос 2:

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

In [10]:
```
question = "I love neural networks"
embeddings = wv_embeddings
features = question_to_vec(question, embeddings, tokenizer)
features[2].round(2)
```

Out[10]: -1.29

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

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

DCG@K

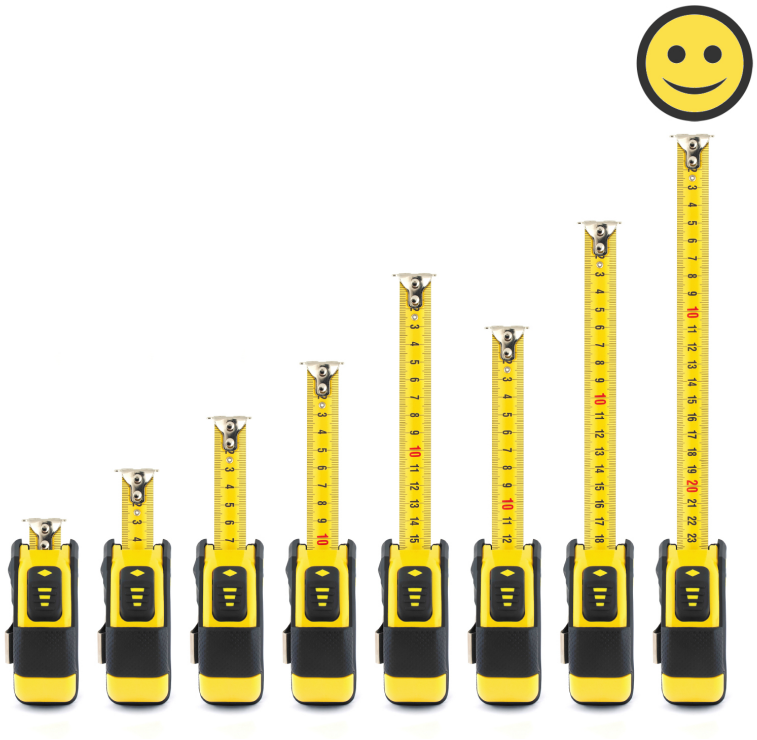
Второй метрикой будет упрощенная DCG метрика, учитывающая порядок элементов в списке путем домножения релевантности элемента на вес равный обратному логарифму номера позиции:: $$ \text{DCG@K} = \frac{1}{N} \sum_{i=1}^N\frac{1}{\log_2(1+rank\_q_i^{'})}\cdot[rank\_q_i^{'} \le K],$$ С такой метрикой модель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

Вопрос 3:

- **Максимум `Hits@47 - DCG@1` ?**

Ответ 3:

1



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] $\text{Hits@1} = [rank\_q_i^{'} \le 1)] = 0$**
- **[K = 4] $\text{Hits@4} = [rank\_q_i^{'} \le 4] = 1$**

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 4:

- **Вычислите `DCG@10` , если $rank\_q_i^{'} = 9$(округлите до одного знака после запятой)**

In [18]:
```
from math import log2
```

In [11]:
```
def dcg(rank, k):
    return 1 / log2(1 + rank) * (rank <= k)

round(dcg(9, k=10), 1)
```

Out[11]:
```
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.

In [4]:
```
def hits_count(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        result: вернуть  Hits@k
    """
    hits_value = np.mean([(rank <= k) for rank in dup_ranks])
    return hits_value
```

In [5]:
```
def dcg_score(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        result: вернуть DCG@k
    """
    dcg_value = np.mean([1 / log2(1 + rank) * (rank <= k)
                         for rank in dup_ranks])
    return dcg_value
```

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

In [6]:
```
import pandas as pd
```

In [15]:
```
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

In [16]:
```
# 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
```

Out[16]:

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

Данные

arxiv link

`train.tsv` **- выборка для обучения.**
В каждой строке через табуляцию записаны: <вопрос>, <похожий вопрос>

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

In [17]:
```
!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.

In [7]:
```
def read_corpus(filename):
    data = []
    for line in open(filename, encoding='utf-8'):
        data.append(line.split("\t"))
    return data
```

Нам понадобиться только файл validation.

In [8]:
```
validation_data = read_corpus('./data/validation.tsv')
```

Кол-во строк

In [9]:
```
len(validation_data)
```

Out[9]: 3760

Размер нескольких первых строк

In [49]:
```
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)].

In [10]:
```
from sklearn.metrics.pairwise import cosine_similarity
from copy import deepcopy
```

In [11]:
```
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)]
    """
    question_vec = question_to_vec(question, embeddings, tokenizer)
    candidates_vecs = [question_to_vec(candidate, embeddings, tokenizer) for candidate in candidates]
    cosines = [cosine_similarity(question_vec[None, :], candidate_vec[None, :]).item() for candidate_vec in candidates_ve
    result = np.array(list(zip(range(len(candidates)), candidates)))
    sorted_idx = np.argsort(-np.array(cosines))
    result = result[sorted_idx]
    return result
```

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

In [52]:
```
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']]
```

In [53]:
```
for question, q_candidates in zip(questions, candidates):
        ranks = rank_candidates(question, q_candidates, wv_embeddings, tokenizer)
        print(ranks)
        print()
```

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

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

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

In [74]:
```
# # 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 5:

102

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

In [12]:
```
from tqdm.notebook import tqdm
```

In [55]:
```
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)
```

In [56]:
```
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)))
```

```
DCG@   1: 0.285 | Hits@   1: 0.285
DCG@   5: 0.342 | Hits@   5: 0.393
DCG@  10: 0.360 | Hits@  10: 0.449
DCG@ 100: 0.406 | Hits@ 100: 0.679
DCG@ 500: 0.431 | Hits@ 500: 0.878
DCG@1000: 0.444 | Hits@1000: 1.000
```

Эмбеддинги, обученные на корпусе похожих вопросов

In [13]:
```
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.

Выбор window

Если взять слишком маленькое окно (например - 2-5), то не весь значимый контекст слова туда попадет. В результате будет много пересечений в словах, которые разные по смыслу, особенно будем много пересечений в антонимах. Если взять большой размер окна (30-50), то в контекст может попасть много лишних слов, которые к контексту конкретного слова особенно и не относятся. И вообще говоря, это гиперпараметр, который выбирают исходя из метрики.

Перебрав варианты [5, 10, 15, 25, 35], исходя из максимизации метрики (см. ниже некоторые результаты метрик) остановился на 35

In [26]:
```
'''your code'''
# Вариант 1 - простой
words = [tokenizer.tokenize(f"{pair[0]} {pair[1]}") for pair in train_data]
```

In [14]:
```
'''your code'''
# Вариант 2 - более сложный

import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stopWords = set(stopwords.words('english'))
nltk.download('wordnet')
wnl = nltk.WordNetLemmatizer()

def preproc_nltk(text):
    return ' '.join([wnl.lemmatize(word) for word in word_tokenize(text) if word not in stopWords])

words = [preproc_nltk(f"{pair[0]} {pair[1]}").split() for pair in train_data]
```

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

In [27]:
```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(words, # data for model to train on
                 size=200,                # embedding vector size
                 min_count=5,            #'''your code'''# consider words that occured at least 5 times
                 window=35).wv            #'''your code'''
```

In [28]:
```
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)
```

In [29]:
```
# window=35, words: Вариант 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)))
```

```
DCG@   1: 0.304 | Hits@   1: 0.304
DCG@   5: 0.378 | Hits@   5: 0.439
DCG@  10: 0.401 | Hits@  10: 0.510
DCG@ 100: 0.450 | Hits@ 100: 0.753
DCG@ 500: 0.471 | Hits@ 500: 0.919
DCG@1000: 0.480 | Hits@1000: 1.000
```

лучшее качество ЗДЕСЬ

In [25]:
```
# window=35, words: Вариант 2
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)))
```

```
DCG@   1: 0.338 | Hits@   1: 0.338
DCG@   5: 0.423 | Hits@   5: 0.499
DCG@  10: 0.449 | Hits@  10: 0.578
DCG@ 100: 0.496 | Hits@ 100: 0.804
DCG@ 500: 0.515 | Hits@ 500: 0.956
DCG@1000: 0.520 | Hits@1000: 1.000
```

In [22]:
```
# window=25, words: Вариант 2
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)))
```

```
DCG@   1: 0.336 | Hits@   1: 0.336
DCG@   5: 0.422 | Hits@   5: 0.499
DCG@  10: 0.447 | Hits@  10: 0.578
DCG@ 100: 0.495 | Hits@ 100: 0.809
DCG@ 500: 0.514 | Hits@ 500: 0.955
DCG@1000: 0.519 | Hits@1000: 1.000
```

In [19]:
```
# window=15, words: Вариант 2
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)))
```

```
DCG@   1: 0.324 | Hits@   1: 0.324
DCG@   5: 0.412 | Hits@   5: 0.491
DCG@  10: 0.438 | Hits@  10: 0.570
```

```
DCG@ 100: 0.485 | Hits@ 100: 0.800
DCG@ 500: 0.506 | Hits@ 500: 0.960
DCG@1000: 0.510 | Hits@1000: 1.000
```

In [102…

```
# window=10, words: Вариант 2
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)))
```

```
DCG@   1: 0.316 | Hits@   1: 0.316
DCG@   5: 0.405 | Hits@   5: 0.486
DCG@  10: 0.430 | Hits@  10: 0.564
DCG@ 100: 0.477 | Hits@ 100: 0.793
DCG@ 500: 0.499 | Hits@ 500: 0.958
DCG@1000: 0.503 | 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.

- **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?**
  - **Лучшее качество дает токенизация с исключением пунктуации и стоп-слов. Это помогает избежать шума, вызванного ими**
- **Помогает ли нормализация слов?**
  - **Нормализация помогает, так как одни и те же слова в разной форме имеют схожий смысл и логичнее их считать одинаковыми. При этом отказ от lowercase результат улучшил. Возможно, выделение первых слов предложения и каких-то названий привело к улучшению метрик**

- **Какие эмбеддинги лучше справляются с задачей и почему?**
  - **эмбединги с бОльшей размерностью работают лучше, поскольку могут учесть в себе больше информации**
- **Почему получилось плохое качество решения задачи?**
  - **Вариантов ответа много. Мои варианты:**
    - **не учитывается порядок слов**
    - **не учитывается смысловая взаимосвязь между словами**
    - **не учитываются n-граммы**
- **Предложите свой подход к решению задачи.**
  - **если без использования нейронок, то я бы предложил:**
    - **использовать n-граммы**
    - **провести подбор гиперпараметров с использованием целевой метрики**
    - **увеличить размерность эмбедингов**

In [ ]: