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

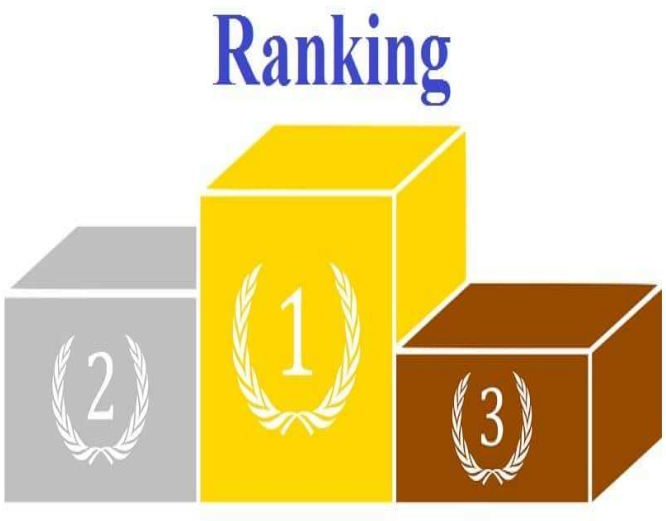
Задача ранжирования(Learning to Rank)

- 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
```

```
--2024-02-27 08:44:15--  https://zenodo.org/record/1199620/files/SO_vectors_200.bin?download=1
Resolving zenodo.org (zenodo.org)... 188.184.103.159, 188.185.79.172, 188.184.98.238, ...
Connecting to zenodo.org (zenodo.org)|188.184.103.159|:443... connected.
HTTP request sent, awaiting response... 301 MOVED PERMANENTLY
Location: /records/1199620/files/SO_vectors_200.bin [following]
--2024-02-27 08:44:15--  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  4.64MB/s    in 4m 24s

2024-02-27 08:48:40 (5.25 MB/s) - ‘SO_vectors_200.bin?download=1’ saved [1453905423/1453905423]
```

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

```
# method most_simmilar
print(f"Top 5 closest words to 'dog': {wv_embeddings.most_similar(positive='dog', topn=5)}")
```

```
print("No, 'cat' not in top 5 for dog.")
print(f"Cat's position is {wv_embeddings.rank('dog', 'cat')}")
```

```
Top 5 closest words to 'dog': [('animal', 0.8564180135726929), ('dogs', 0.7880866527557373), ('mammal', 0.7623804211616516), ('cats
No, 'cat' not in top 5 for dog.
Cat's position is 26
```

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

```
def question_to_vec(question:str, embeddings:list[int], tokenizer, dim:int=200, pre_norm:bool=False, post_norm:bool=False) -> list[int]:
    """
        question: строка
        embeddings: наше векторное представление
        dim: размер любого вектора в нашем представлении

        return: векторное представление для вопроса
    """
    tokens = tokenizer().tokenize(question.lower())

    return embeddings.get_mean_vector(keys=tokens, pre_normalize=pre_norm, post_normalize=post_norm, ignore_missing=True)[:dim]
```

```
print(question_to_vec('How to use Gensim and NLTK for NLP tasks in Google Colab?', wv_embeddings, WordPunctTokenizer))
```

```
[ 0.51679343  0.12172341  0.35998496 -0.4319114  -1.0370318   0.9299659
  1.2020712  -0.36596173  0.44069594  0.15495043 -1.1228906  -0.8447539
 -0.41401604 -0.02108631  1.4079322   0.36768296 -0.42918468  0.30979016
 -0.549481   -0.3993114   0.5301982   1.5252502  -0.95741326 -0.17516652
  0.32587686 -0.7046796  -0.17205478  0.59909433  0.26953658 -0.90959615
 -1.3667134   0.32744414 -1.1641706  -0.8531855   0.7451635   0.44581035
 -0.4981935  -0.66160554  0.98003095  0.46799737 -0.408206   -0.10371072
  1.0035719  -0.28068486  0.16196837  0.08332462 -1.7079794   0.90424365
  0.9915622  -0.09759611  1.2851696   0.18007715 -0.99973404 -1.5195916
 -0.9983058   0.5622769   0.23195772  0.6816965  -0.6364785  -0.2540265
 -0.28495544 -0.1210212  -0.38199216 -0.8660377   0.50349134 -1.6254216
  1.2122718   0.6154348  -0.16097412 -1.1069462   0.9343215  -0.84239435
  0.45441464  0.28014034 -0.6930428  -2.6844242   0.10364325 -0.30110797
  0.091696   -0.61248046 -0.18508728 -0.25780526  0.88650614 -0.264114
  0.40808982 -0.24961092 -1.207342    0.45091408 -0.29941633 -0.39394504
 -0.1495786  -0.6421443   0.6232456  -0.71312726  0.8933191   0.6600588
  0.19069047  0.26539534 -0.6270256   1.0214566   0.1942047  -0.30791473
 -0.17224999 -0.06592895 -0.778336   -0.18394212  0.07408642  0.08943915
  0.44687027  1.395112    0.52625173 -0.5940265  -0.4578216   0.9698982
  0.51243716 -0.57084763 -1.5766561  -0.33469334  0.717171   -0.06568348
  0.43957165 -0.7317742  -1.1494293   0.48356816 -0.33885035  0.2622783
  0.28580981 -1.954369   -0.1908896   0.09776188  0.04591533 -1.6649078
  0.39837733  1.7146227   0.21335292 -0.6799913   1.5455787  -0.34915537
  0.2331395  -0.33886215  0.35233578 -0.6565318  -0.790383    0.30992284
 -0.29745594 -1.0240222  -0.45146102 -2.0135553   0.04023749  0.5703616
 -0.9120246  -0.07062328 -0.3105317   0.79175717  0.8899488  -1.236714
 -1.1055716  -0.16588123  0.6111773   0.23018335  0.98721534  0.4047863
 -0.86345     0.1085611  -0.7006878  -0.13337885  1.450577   -1.0693009
 -1.1895106  -0.593689    0.6923974   1.4869244  -0.38438     1.752474
  0.39914075 -1.0305151  -1.1981212   1.5572846   1.1297306   0.28872818
  0.1471818   0.9975357  -1.7963667   0.4442722  -0.45830506  1.1615006
  1.105139   -0.27003306  0.5254023  -0.3905159  -0.84541005 -1.0423003
 -0.9277226   0.26477826 -1.9250143  -0.9248409  -0.09050813  0.05733122
  0.3193057   0.7317912 ]
```

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)?

```
'''your code'''
tokenizer = WordPunctTokenizer()
tokens = tokenizer.tokenize('I love neural networks')
print(f'Третья компонента равна = {wv_embeddings.get_mean_vector(tokens, pre_normalize=False, ignore_missing=True)[2]:.2f}')
```

```
Третья компонента равна = -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:

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

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

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

Вопрос 3:

- Максимум `Hits@47 - DCG@1`?

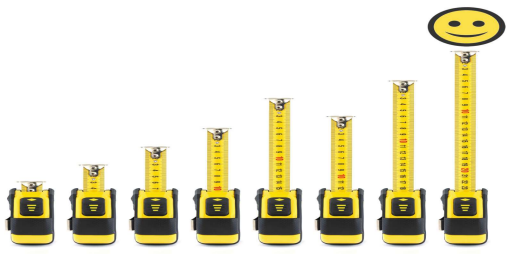
$DCG@1 \in [0, 1]$

$Hits@47 \in [0, 1]$

Однако разность `Hits@47` и `DCG@1` не может быть меньше 0, поскольку у нас не может быть DCG = 1 и Hits = 0. Следовательно

$Hits@47 - DCG@1 \in [0, 1]$

При $rank_q$ = 1 мы получаем 0, при $rank_q \in [2, 47]$ мы получаем 1 (максимум) и при $rank_q \geq 48$ мы получаем 0



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

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

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

Вопрос 4:

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

$$DCG@10 = \frac{1}{\log_2(1+9)} * [9 \leq 10] = 0.3$$

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: list[int], k:int) -> int:
    """
        dup_ranks: list индексов дубликатов
        result: вернуть  Hits@k
    """
    '''your code'''
    dup_ranks_np = np.array(dup_ranks)
    hits_value = (dup_ranks_np <= k).mean()
    return hits_value


def dcg_score(dup_ranks: list[int], k:int) -> int:
    """
        dup_ranks: list индексов дубликатов
        result: вернуть DCG@k
    """
    '''your code'''
    dup_ranks_np = np.array(dup_ranks)
    activation = dup_ranks_np <= k
    dcg_value = ((1/np.log2(1 + dup_ranks_np)) * activation).mean()

    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 was thrown",]

# наги 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

Next steps: View recommended plots

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>, ...**

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

```
Mounted at /content/drive
```

```
!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:str) -> list[str]:
    data = []
    for line in open(filename, encoding='utf-8'):
        '''your code'''
        data.append(line.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(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: str, candidates: list[str], embeddings: list[int], tokenizer, dim:int=200, pre_norm: bool = False, post_n
    """
        question: строка
        candidates: массив строк(кандидатов) [a, b, c]
        result: пары (начальная позиция, кандидат) [(2, c), (0, a), (1, b)]
    """
    sim = cosine_similarity([question_to_vec(question, embeddings, tokenizer, dim=dim, pre_norm=pre_norm, post_norm=post_norm)], [questi
    result = list(zip(sim, range(len(candidates)), candidates))
    result.sort(reverse = True)
    return [res[1:] for res in 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, WordPunctTokenizer)
        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 j

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

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?')],
           [(0, 'Getting all list items of an unordered list in PHP'),     #скрыт
            (2, 'select2 not displaying search results'),                  #скрыт
            (1, '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, WordPunctTokenizer)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

```
  27%                                                1000/3760 [02:00<17:58,  2.56it/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, 76.45it/s]
DCG@   1: 0.397 | Hits@   1: 0.397
DCG@   5: 0.494 | Hits@   5: 0.578
DCG@  10: 0.516 | Hits@  10: 0.644
DCG@ 100: 0.563 | Hits@ 100: 0.875
DCG@ 500: 0.575 | Hits@ 500: 0.973
DCG@1000: 0.578 | Hits@1000: 1.000
```

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.

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

Baseline

```
words = [tokenizer.tokenize(' '.join(question).lower()) for question in train_data]
```

```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(words,              # data for model to train on
                 vector_size=200,                 # embedding vector size
                 min_count=5,                     # consider words that occured at least 5 times
                 window=5).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, WordPunctTokenizer)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

```
 27%                                                 1000/3760 [02:41<06:26,  7.14it/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, 134.78it/s]
DCG@   1: 0.287 | Hits@   1: 0.287
DCG@   5: 0.368 | Hits@   5: 0.438
DCG@  10: 0.391 | Hits@  10: 0.510
DCG@ 100: 0.446 | Hits@ 100: 0.780
DCG@ 500: 0.468 | Hits@ 500: 0.952
DCG@1000: 0.473 | Hits@1000: 1.000
```

Tokenizers

ToktokTokenizer

```
from nltk.tokenize import ToktokTokenizer

tokenizer = ToktokTokenizer()


words = [tokenizer.tokenize(' '.join(question).lower()) for question in train_data]


from gensim.models import Word2Vec
embeddings_trained = Word2Vec(words,               # data for model to train on
                 vector_size=200,                  # embedding vector size
                 min_count=5,                      # consider words that occured at least 5 times
                 window=5).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, ToktokTokenizer, 200)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

```
 27%                                                 1000/3760 [03:12<07:28,  6.15it/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, 122.76it/s]
DCG@   1: 0.277 | Hits@   1: 0.277
DCG@   5: 0.356 | Hits@   5: 0.427
DCG@  10: 0.378 | Hits@  10: 0.496
DCG@ 100: 0.428 | Hits@ 100: 0.742
DCG@ 500: 0.453 | Hits@ 500: 0.935
DCG@1000: 0.460 | Hits@1000: 1.000
```

TREE

```
from nltk.tokenize import TreebankWordTokenizer

tokenizer = TreebankWordTokenizer()


words = [tokenizer.tokenize(' '.join(question).lower()) for question in train_data]


from gensim.models import Word2Vec
embeddings_trained = Word2Vec(words,               # data for model to train on
                 vector_size=200,                  # embedding vector size
                 min_count=5,                      # consider words that occured at least 5 times
                 window=5).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, TreebankWordTokenizer, 200)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

```
  27%                                          1000/3760 [03:55<08:15,  5.57it/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, 120.47it/s]
DCG@   1: 0.282 | Hits@   1: 0.282
DCG@   5: 0.362 | Hits@   5: 0.432
DCG@  10: 0.380 | Hits@  10: 0.487
DCG@ 100: 0.435 | Hits@ 100: 0.753
DCG@ 500: 0.459 | Hits@ 500: 0.942
DCG@1000: 0.465 | Hits@1000: 1.000
```

Normalization

Pre_norm

```
tokenizer = WordPunctTokenizer()
```

```
words = [tokenizer.tokenize(' '.join(question).lower()) for question in train_data]
```

```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(words,               # data for model to train on
                 vector_size=200,                  # embedding vector size
                 min_count=5,                      # consider words that occured at least 5 times
                 window=5).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, WordPunctTokenizer, 200, True, False)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

```
  27%                                          1000/3760 [02:48<06:42,  6.86it/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, 108.12it/s]
DCG@   1: 0.320 | Hits@   1: 0.320
DCG@   5: 0.404 | Hits@   5: 0.477
DCG@  10: 0.429 | Hits@  10: 0.554
DCG@ 100: 0.478 | Hits@ 100: 0.797
DCG@ 500: 0.499 | Hits@ 500: 0.959
DCG@1000: 0.503 | Hits@1000: 1.000
```

Post Norm

```
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, WordPunctTokenizer, 200, False, True)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

```
  27%                                          1000/3760 [02:36<06:28,  7.10it/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, 152.72it/s]
DCG@   1: 0.289 | Hits@   1: 0.289
DCG@   5: 0.368 | Hits@   5: 0.436
DCG@  10: 0.394 | Hits@  10: 0.516
DCG@ 100: 0.447 | Hits@ 100: 0.777
DCG@ 500: 0.469 | Hits@ 500: 0.949
DCG@1000: 0.474 | Hits@1000: 1.000
```

Pre and Post Norm

```
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, WordPunctTokenizer, 200, True, True)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

```
  27%                                          1000/3760 [03:03<07:00,  6.56it/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, 143.19it/s]
DCG@   1: 0.320 | Hits@   1: 0.320
DCG@   5: 0.404 | Hits@   5: 0.477
DCG@  10: 0.429 | Hits@  10: 0.554
DCG@ 100: 0.478 | Hits@ 100: 0.797
DCG@ 500: 0.499 | Hits@ 500: 0.959
DCG@1000: 0.503 | Hits@1000: 1.000
```

Best

```
import string
from gensim.parsing.preprocessing import STOPWORDS

stop_words = STOPWORDS.union(string.punctuation)
tokenizer = WordPunctTokenizer()


words = [[word for word in tokenizer.tokenize(' '.join(sent).lower()) if word not in stop_words] for sent in train_data]
```

```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(words,               # data for model to train on
                 vector_size=200,                  # embedding vector size
                 min_count=10,                     # consider words that occured at least 5 times
                 window=25).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, WordPunctTokenizer, 200, True)
    wv_ranking.append([r[0] for r in ranks].index(0) + 1)
```

```
  27%                                          1000/3760 [02:15<05:22,  8.57it/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, 106.57it/s]
DCG@   1: 0.475 | Hits@   1: 0.475
DCG@   5: 0.576 | Hits@   5: 0.664
DCG@  10: 0.596 | Hits@  10: 0.727
DCG@ 100: 0.635 | Hits@ 100: 0.915
DCG@ 500: 0.644 | Hits@ 500: 0.984
DCG@1000: 0.646 | Hits@1000: 1.000
```

Вывод:

- С baseline лучше всего из протестированных справился WordPunctTokenizer. Возможно причина лежит в том, что такой простой способ токенизации на нашем датасете не плодит лишние слова, что позволяет уместить больше сути в окна, тем самым повысив точность модели по нашим метрикам.
- Пре нормализация улучшает наши результаты значительно, пост нормализация немного тоже помогает, но использовать оба смысла нет, так как тогда результат будет по метрикам как от пре нормализации.
- Если брать только Baseline, то лучше справляются предобученные из Gensim. Скорее всего дело в большом количестве данных для обучения. Если брать лучший результат, то лучше наше решение, потому что оно больше подогнано под нашу задачу.
- Маленькое окно захватывало слишком мало контекста + стоп слова мешали захвату контекста, потому что занимали место в окне.
- Убрать стоп-слова, сделать окно в районе 25 и min_count в районе 10, сделать пре-нормализацию векторов. Изменять длину векторов особо не помогает.