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

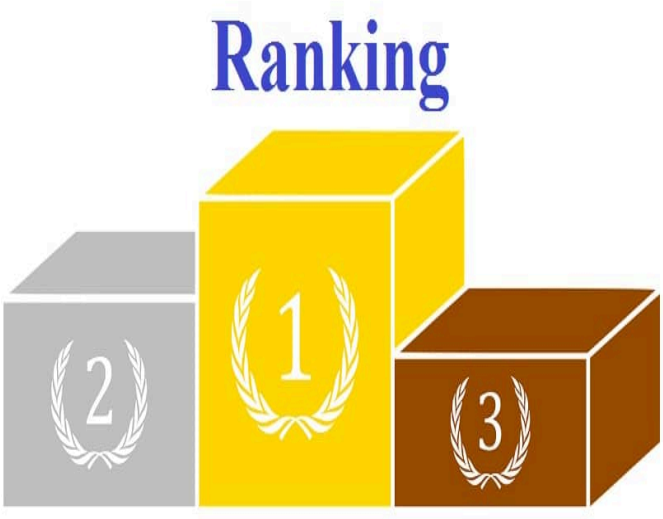
На обучающей выборке задан порядок между некоторыми элементами, то есть нам известно, что некий объект выборки более релевантный для нас, чем другой:

- $i < j$ - порядок пары индексов объектов на выборке X^l с индексами i и j

Задача:

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

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



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
```

```
#!pip install -q gensim
```

```
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
```

```
wv_embeddings.vectors.shape
```

```
(1787145, 200)
```

Найдем наиболее близкие слова к слову `dog`:

Вопрос 1:

- Входит ли слово `cat` в топ-5 близких слов к слову `dog`? Какое место оно занимает?

```
# method most_simmilar

similar_words = wv_embeddings.most_similar('dog', topn=5)
similar_words
```

```
[('animal', 0.8564180135726929),
 ('dogs', 0.7880866527557373),
 ('mammal', 0.7623804211616516),
 ('cats', 0.7621253728866577),
 ('animals', 0.760793924331665)]
```

```
similarity_score = wv_embeddings.similarity('dog', 'cat')
print(f"Similarity between 'dog' and 'cat': {similarity_score:.4f}")
```

```
Similarity between 'dog' and 'cat': 0.6852
```

```
similar_words = wv_embeddings.most_similar('dog', topn=100)
similar_words[-1]

# ('foxes', 0.5991548895835876) - занимает 100-е место, значит 'cat' входит в сотню, найдем точную позицию
```

```
('foxes', 0.5991548895835876)
```

```
for i, word in enumerate(similar_words):
  if ('cat' == word[0]):
    print(f'{word} has index {i}')
```

```
('cat', 0.6852341294288635) has index 25
```

Ваш ответ: 'cat' не входит в топ-5 ближайших слов к 'dog'. 'cat' занимает 26-е место

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-3009405.py:9: SyntaxWarning: invalid escape sequence '\w'
  return re.findall('\w+', text)
```

```
from nltk.tokenize import WordPunctTokenizer

tokenizer = WordPunctTokenizer()
```

```
import numpy as np

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
    """

    features = np.zeros([dim], dtype='float32')

    i = 0
    for word in tokenizer.tokenize(question):
        if word in embeddings:
            i += 1
            features += embeddings[f'{word}']

    return features / i if i > 0 else 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)?

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

vec = question_to_vec(question, wv_embeddings, tokenizer)

print(f"Третья компонента вектора: {vec[2]:.2f}")
```

```
Третья компонента вектора: -1.29
```

Ответ: Третья компонента вектора: -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:

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

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

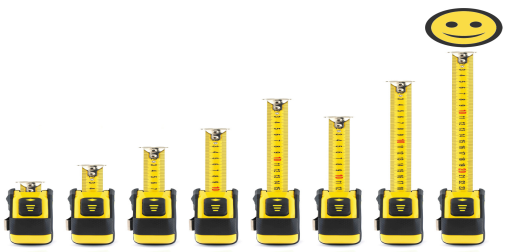
DCG@K

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

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

С такой метрикой модель штрафуется за большой ранк корректного ответа.

DCG@K измеряет качество ранжирования, учитывая не только факт наличия правильного ответа в топ-K, но и ***его точную позицию***.



Пример оценок

Вычислим описанные выше метрики для игрушечного примера. Пусть

- $N = 1, R = 3$
- "Что такое python?" - вопрос q_1
- "Что такое язык python?" - его дубликат q_i'

Пусть модель выдала следующий ранжированный список кандидатов:

1. "Как изучить c++?"
2. "Что такое язык python?"
3. "Хочу учить Java"
4. "Не понимаю Tensorflow"

$\Rightarrow rank\_q_i' = 2$

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

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

Проверяем условие $\text{rank}_{q_1'} \leq 1$: ***условие неверно***.

Следовательно, $[\text{rank}_{q_1'} \leq 1] = 0$.

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

Проверяем условие $\text{rank}_{q_1'} \leq 4$: ***условие верно***.

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

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

```
import math

dcg_10 = 1 / math.log2(1 + 9) * 1
print(f'DCG@10 = {dcg_10:.1f}')
```

```
DCG@10 = 0.3
```

Ответ: DCG@10 = 1 / log2(10) = 0.3

Более сложный пример оценок

Рассмотрим пример с $N > 1$, где $N = 3$ (три вопроса) и для каждого вопроса заданы позиции их дубликатов. Вычислим метрики **Hits@K** для разных значений K.

- $N = 3$: Три вопроса (q_1, q_2, q_3).
- Для каждого вопроса известна позиция его дубликата ($\text{rank}_{q_i'}$):
  - $\text{rank}_{q_1'} = 2$,
  - $\text{rank}_{q_2'} = 5$,
  - $\text{rank}_{q_3'} = 1$.

Мы будем вычислять **Hits@K** для $K = 1, 5$.

Для $K = 1$:

Подставим значения:

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

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

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

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:

$$\text{Hits@5} = \frac{1}{3} \cdot \left([\text{rank}_{q_1'} \le 5] + [\text{rank}_{q_2'} \le 5] + [\text{rank}_{q_3'} \le 5] \right).$$

Проверяем условие $\text{rank}_{q_i'} \le 5$ для каждого вопроса:

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

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:

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

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

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

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:

$$\text{DCG@5} = \frac{1}{3} \cdot \left(\frac{1}{\log_2(1 + \text{rank}_{q_1'})} \cdot [\text{rank}_{q_1'} \le 5] + \frac{1}{\log_2(1 + \text{rank}_{q_2'})} \cdot [\text{rank}_{q_2'} \le 5] + \frac{1}{\log_2(1 + \text{rank}_{q_3'})} \cdot [\text{rank}_{q_3'} \le 5] \right).$$

Проверяем условие $\text{rank}_{q_i'} \le 5$ для каждого вопроса:

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

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@47 = Hits@5 = 1
- DCG@1 = 1/3

Ответ: Hits@47 - DCG@1 = 2/3

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).

К примеру для "Что такое язык python?" $dup\_ranks = [2]$.

```
def hits_count(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        k: пороговое значение для ранга
        result: вернуть Hits@k
    """
    # Подсчитываем количество дубликатов, чей ранг <= k

    N = len(dup_ranks)
    sum = 0
    for dup in dup_ranks:
      if dup <= k:
        sum += 1

    return sum / 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
    """

    N = len(dup_ranks)
    sum = 0
    for dup in dup_ranks:
      if dup <= k:
        sum += 1 / math.log2(1 + dup)

    return sum / N
```

```
# 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

```
import pandas as pd

# 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: Generate code with correct_answers | 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>, ...**

```
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_
```

```
%cd /content/drive/My Drive
```

```
/content/drive/My Drive
```

```
#!unzip stackoverflow_similar_questions.zip
```

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
```

```
validation_data[0][0:3]
```

```
['How to print a binary heap tree without recursion?',
 'How do you best convert a recursive function to an iterative one?',
 'How can i use ng-model with directive in angular js']
```

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)]
    """

    question_vec = question_to_vec(question, embeddings, tokenizer)
    candidates_vecs = [(question_to_vec(c, embeddings, tokenizer), c) for c in candidates]

    question_vec_r = question_vec.reshape(1, -1)
    simularities = []
    for i, c in enumerate(candidates_vecs):
      c_r = c[0].reshape(1, -1)
      simularities.append((i, cosine_similarity(question_vec_r, c_r)[0][0], c[1]))

    sorted_simularities = sorted(simularities, key=lambda x: x[1], reverse=True)

    return [(c[0], c[2]) for c in sorted_simularities]
```

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'),

[(1, 'WPF- How to update the changes in list item of a list'), (0, 'Getting all list items of an unordered list in P
```

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?')],
           [(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')]]
```

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: 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.

```
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 [09:18<27:53,  1.65it/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)))
```

```
100%  6/6 [00:00<00:00, 237.98it/s]
DCG@   1: 0.276 | Hits@   1: 0.276
DCG@   5: 0.332 | Hits@   5: 0.386
DCG@  10: 0.349 | Hits@  10: 0.438
DCG@ 100: 0.395 | Hits@ 100: 0.671
DCG@ 500: 0.421 | Hits@ 500: 0.875
DCG@1000: 0.434 | 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 $\frac{1}{\log_2(1+\mathrm{rank}_{q'})}$.

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']
```

```
print(train_data[111258][0])
print(train_data[111258][1])
print(train_data[111258][0] + ' ' +train_data[111258][1])
print(tokenizer.tokenize(train_data[111258][0] + ' ' +train_data[111258][1]))
```

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

```
import nltk

from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords

nltk.download('punkt_tab')
nltk.download('wordnet')
nltk.download('stopwords')

english_stop_words = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab 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!
```

```
from nltk.tokenize import WordPunctTokenizer

class NltkWithStopWordsTokenizer:
    def __init__(self):
        self.word_punct_tokenizer = WordPunctTokenizer()

    def tokenize(self, text):
        return [word for word in self.word_punct_tokenizer.tokenize(text.lower()) if word not in english_stop_words

tokenizer = NltkWithStopWordsTokenizer()
```

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

for i, qa in enumerate(tqdm(train_data)):
  line = qa[0] + ' ' + qa[1]
  corpus.append(tokenizer.tokenize(line))
```

```
100%                                                1000000/1000000 [00:17<00:00, 50795.32it/s]
```

```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(
    sentences=corpus,        # Корпус токенизированных текстов
    vector_size=200,         # Размерность векторов
    window=10,               # Размер окна контекста
    min_count=3,             # Минимальная частота слов
    workers=4                # Количество потоков
).wv
```

Выбираем размер окна 10, чтобы обучалось достататочно быстро. Это значение больше дефолтного значения 5, чтобы захватить больше контекста, так у нас задача ранжинования.

```
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 [08:38<21:32,  2.14it/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, 364.61it/s]
DCG@   1: 0.411 | Hits@   1: 0.411
DCG@   5: 0.496 | Hits@   5: 0.572
DCG@  10: 0.518 | Hits@  10: 0.641
DCG@ 100: 0.566 | Hits@ 100: 0.869
DCG@ 500: 0.580 | Hits@ 500: 0.973
DCG@1000: 0.583 | Hits@1000: 1.000
```

nltk + lemmatizer

```
import nltk

from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords

nltk.download('punkt_tab')
nltk.download('wordnet')
nltk.download('stopwords')

english_stop_words = set(stopwords.words('english'))

wnl = nltk.WordNetLemmatizer()
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab 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!
```

```
class NltkLemmWithStopWordsTokenizer:
    def __init__(self):
        self.word_punct_tokenizer = WordPunctTokenizer()

    def tokenize(self, text):
        return [wnl.lemmatize(word) for word in self.word_punct_tokenizer.tokenize(text.lower()) if word not in eng

tokenizer = NltkLemmWithStopWordsTokenizer()
```

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

for i, qa in enumerate(tqdm(train_data)):
  line = qa[0] + ' ' + qa[1]
  corpus.append(tokenizer.tokenize(line))
```

```
100%  1000000/1000000 [01:28<00:00, 13003.49it/s]
```

```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(
    sentences=corpus,        # Корпус токенизированных текстов
    vector_size=200,         # Размерность векторов
    window=10,               # Размер окна контекста
    min_count=3,             # Минимальная частота слов
    workers=4                # Количество потоков
).wv
```

Выбираем размер окна 10, чтобы обучалось достататочно быстро. Это значение больше дефолтного значения 5, чтобы захватить больше контекста, так у нас задача ранжинования.

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

for k in tqdm([1, 5, 10, 100, 500, 1000]):
    print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg_score(wv_ranking, k), k, hits_count(wv_ranking, k)))
```

```
 27%                                          1000/3760 [09:21<22:13,  2.07it/s]

100%                                          6/6 [00:00<00:00, 211.57it/s]
DCG@   1: 0.425 | Hits@   1: 0.425
DCG@   5: 0.510 | Hits@   5: 0.587
DCG@  10: 0.532 | Hits@  10: 0.656
DCG@ 100: 0.577 | Hits@ 100: 0.876
DCG@ 500: 0.590 | Hits@ 500: 0.973
DCG@1000: 0.593 | Hits@1000: 1.000
```

Spacy (not finished - long to train)

```
import spacy
nlp = spacy.load('en_core_web_sm')
```

```
class NltkLemmWithStopWordsTokenizer:
    def __init__(self):
        self.word_punct_tokenizer = WordPunctTokenizer()

    def tokenize(self, text):
        spacy_results = nlp(text)
        return [token.lemma_ for token in spacy_results if token.lemma_ not in english_stop_words]

tokenizer = NltkLemmWithStopWordsTokenizer()
```

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

for i, qa in enumerate(tqdm(train_data)):
  line = qa[0] + ' ' + qa[1]
  corpus.append(tokenizer.tokenize(line))
```

```
 12%                                          117882/1000000 [22:46<3:34:47, 68.45it/s]
```

```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(
    sentences=corpus,        # Корпус токенизированных текстов
    vector_size=200,         # Размерность векторов
    window=10,                # Размер окна контекста
    min_count=3,             # Минимальная частота слов
    workers=4                # Количество потоков
).wv
```

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

```
for k in tqdm([1, 5, 10, 100, 500, 1000]):
    print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg_score(wv_ranking, k), k, hits_count(wv_ranking, k)))
```

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:

- nltk tokenizer + stop words немного хуже чем nltk tokenizer + stop words + lemmatization за счет нормализации - получилось больше одинаковых слов на маленьком объеме текста (уменьшился словарь)
- нормализация помогла не сильно в данном случае. Нормализация, теоретически, должна помочь для задачи поиска схожих текстов, так номализация убирает окончания слов и создает одинаковые слова ("buy", "buys", "buying" -> "buy")
- обученный на train датасете эмеддинги дали лучший результат по сравнению со скаченным эмбедингами, т.к. в train datasete текст похож на валидационный
- Качество могло получиться плохое из-за размера эмбеддинга и контекстного окна
- Можно попытаться улучшить качество увеличив контекстное окно и размер эмбеддинга или изменить алгоритм создания