Deep Learning School

Физтех-Школа Прикладной математики и информатики (ФПМИ) МФТИ

Some parts of the notebook are almost the copy of [mmta-team course](). Special thanks to mmta-team for making them publicly available. [Original notebook]().

Прочитайте семинар, пожалуйста, для успешного выполнения домашнего задания. В конце ноутка напишите свой вывод. Работа без вывода оценивается ниже.

Задача поиска схожих по смыслу предложений

Мы будем ранжировать вопросы StackOverflow на основе семантического векторного представления

До этого в курсе не было речи про задачу ранжировния, поэтому введем математическую формулировку

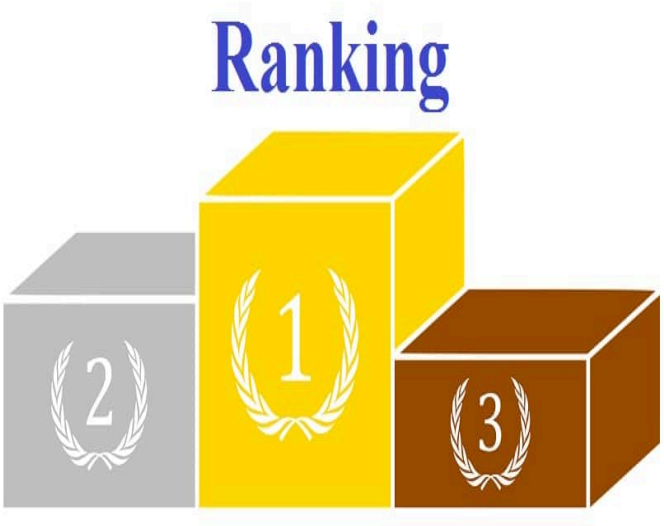
Задача ранжирования(Learning to Rank)

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

Задача:

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

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



Embeddings

Будем использовать предобученные векторные представления слов на постах Stack Overflow.
[A word2vec model trained on Stack Overflow posts]

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

```
--2024-10-10 16:54:13--  https://zenodo.org/record/1199620/files/SO_vectors_200.bin?download=1
Resolving zenodo.org (zenodo.org)... 188.185.79.172, 188.184.103.159, 188.184.98.238, ...
Connecting to zenodo.org (zenodo.org)|188.185.79.172|:443... connected.
HTTP request sent, awaiting response... 301 MOVED PERMANENTLY
Location: /records/1199620/files/SO_vectors_200.bin [following]
--2024-10-10 16:54:14--  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  21.9MB/s    in 67s

2024-10-10 16:55:21 (20.8 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
wv_embeddings.most_similar('dog')
```

```
[('animal', 0.8564180135726929),
 ('dogs', 0.7880866527557373),
 ('mammal', 0.7623804211616516),
 ('cats', 0.7621253728866577),
 ('animals', 0.760793924331665),
 ('feline', 0.7392398715019226),
 ('bird', 0.7315488457679749),
 ('animal1', 0.7219215631484985),
 ('doggy', 0.7213349342346191),
 ('labrador', 0.7209131717681885)]
```

```
word = 'cat'
count = 0
other_words = []
for i,w in enumerate(wv_embeddings.most_similar('dog')):
  if w[0] == word and i < 5:
    print(f'Слово {word} входит в топ-5 близких слов к слову dog')
    break
  else:
    count += 1
    if count == 5:
      print(f'Слово {word} НЕ входит в топ-5 близких слов к слову dog')
      break
```

```
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
    """
    count = 0
    q_emb_sum = np.zeros(shape=dim)

    for w in tokenizer.tokenize(question.lower()):
      if w in embeddings:
        q_emb_sum += embeddings[w]
        count += 1
    return q_emb_sum / count if count > 0 else q_emb_sum
```

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)?

```
sentence = 'I love neural networks'
```

```
round(question_to_vec(question=sentence, embeddings=wv_embeddings, tokenizer=tokenizer)[2], 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:

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

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

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

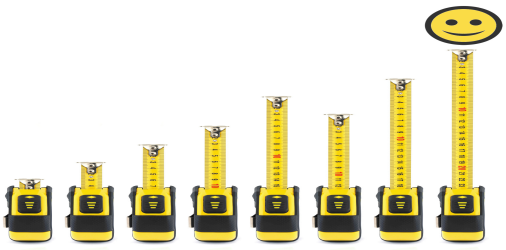
С такой метрикой модель штрафуется за большой ранк корректного ответа

Вопрос 3:

- Максимум `Hits@47 - DCG@1` ?

Ответ: 1

1. Hits@47 = 1 для N=1 во всех случаях, когда rank_q1' дубликата <= 47. В нашем случае rank_q1' = 2.
2. DCG@1 = 0 для N=1: 1 / log2(1 + rank_q1') * [rank_q1' <= 1] = = 1 / log2(3) * [2 <= 1] = 0.
3. Таким образом, во всех случаях, когда rank_q1' принадлежит диапазону (1;47]: максимум (Hits@47 - DCG@1) = 1. При этом в самом "близком" случае rank_q1' = 1 максимум (Hits@47 - DCG@1) = max(1-1) = 0, в этом случае значение двух метрик совпадает.



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)

```
ranq = 9
K = 10
DCG10score = (1 / np.log2(1 + ranq)) * (ranq <= K)
print(f'DCG@10 для rank_q’i=9 равно {round(DCG10score,1)}')
```

```
DCG@10 для rank_q’i=9 равно 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
    """
    hits_value = np.mean(np.array(dup_ranks) <= k)
    return hits_value


def dcg_score(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        result: вернуть DCG@k
    """
    ranks = np.array(dup_ranks)
    flags =  (ranks <= k)
    dcg_value = np.mean(1 / np.log2(1 + ranks) * flags)
    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 = [candidates_ranking[answer].index(copy_answers[answer]) + 1 for answer in range(len(copy_answers))]

# вычисляем метрику для разных k
print('Ваш ответ HIT:', [hits_count(dup_ranks, k) for k in range(1, 5)])
print('Ваш ответ DCG:', [round(dcg_score(dup_ranks, k), 5) for k in range(1, 5)])
```

```
Ваш ответ HIT: [0.0, 1.0, 1.0, 1.0]
Ваш ответ DCG: [0.0, 0.63093, 0.63093, 0.63093]
```

У вас должно получиться

```
# correct_answers - метрика для разных k
correct_answers = pd.DataFrame([[0, 1, 1, 1], [0, 1 / (np.log2(3)), 1 / (np.log2(3)), 1 / (np.log2(3))]],
                               index=['HITS', 'DCG'], columns=range(1,5))
correct_answers
```

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

Далее: Посмотреть рекомендованные графики | New interactive sheet

Данные

arxiv link

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

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

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

```
Archive:  /content/stackoverflow_similar_questions.zip
   creating: data/
  inflating: data/.DS_Store
   creating: __MACOSX/
   creating: __MACOSX/data/
  inflating: __MACOSX/data/._.DS_Store
  inflating: data/train.tsv
  inflating: data/validation.tsv
```

Считайте данные.

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

Нам понадобиться только файл validation.

```
validation_data = read_corpus('./data/validation.tsv')
```

Кол-во строк

```
len(validation_data)
```

```
3760
```

Размер нескольких первых строк

```
for i in range(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_vec = [question_to_vec(question, embeddings, tokenizer, dim)]
    candidates_vec = [question_to_vec(sent, embeddings, tokenizer, dim) \
                      for sent in candidates]
    cosine_sim = np.argsort(cosine_similarity(q_vec, candidates_vec).ravel())[::-1]
    return [(i, candidates[i]) for i in cosine_sim]
```

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,end='')
        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 экперимента 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

Последовательность начальных индексов для эксперимента 2 - `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
with tqdm(desc="Corpus processed", total=max_validation_examples) as pbar_outer:
  for i, line in enumerate(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)
      pbar_outer.update(1)
```

```
Corpus processed: 100%                                  1000/1000 [01:14<00:00, 14.80it/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, 156.59it/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
```

Эмбеддинги, обученные на корпусе похожих вопросов

```
train_data = read_corpus('./data/train.tsv')
```

```
train_data[:7]
```

```
[['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?'],
 ['Sending array via Ajax fails',
  'Getting all list items of an unordered list in PHP'],
 ['How to insert CookieCollection to CookieContainer?',
  'C# create cookie from string and send it'],
 ['Updating one element of a bound Observable collection',
  'WPF- How to update the changes in list item of a list'],
 ['MongoDB error on find()',
  'Retrieve only the queried element in an object array in MongoDB collection'],
 ['select2 not displaying search results',
  'How to use jQuery AJAX for an outside domain?']]
```

Улучшите качество модели.

Склеим вопросы в пары и обучим на них модель Word2Vec из gensim. Выберите размер window. Объясните свой выбор.

```
new_corpus = [couple[0] + str(' ') + couple[1] for couple in train_data]
new_corpus[: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?']
```

Посчитаем минимальную и средние длины предложений в получившемся корпусе

```
corpus_lens = []
for i in tqdm(range(len(new_corpus))):
  corpus_lens.append(len(new_corpus[i].split()))
```

```
100%                                                1000000/1000000 [00:01<00:00, 653026.05it/s]
```

```
corpus_lens[:10]
```

```
[13, 19, 15, 14, 20, 16, 14, 35, 11, 25]
```

```
mean_len = np.mean(corpus_lens)
min_len = min(corpus_lens)
print(f'Средняя длина вопроса - {round(mean_len, 2)} токенов. ')
print(f'Минимальная длина вопроса в корпусе - {min_len} токена. ')
```

```
Средняя длина вопроса - 16.99 токенов.
Минимальная длина вопроса в корпусе - 2 токена.
```

Видим, что после "склейки" средняя длина вопросов стала значительной, при этом до "склейки" в корпусе содержались предложения, состоящие лишь из 1го токена.

```
proc_words = [tokenizer.tokenize(text) for text in tqdm(new_corpus)]
```

```
100%                                                1000000/1000000 [00:14<00:00, 34621.74it/s]
```

```
proc_words[0]
```

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

Исходя из среднего и мин размера вопроса в корпусе, укажем размер окна = 7, при этом размер скрытого слоя укажем равным 300

```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(proc_words, # data for model to train on
                 vector_size=300,                 # embedding vector size
                 min_count=5,             # consider words that occured at least 5 times
                 window=7).wv
```

```
print(embeddings_trained.index_to_key[:5])
```

```
['to', 'in', 'a', 'How', 'the']
```

```
wv_ranking = []
max_validation_examples = 1000
dim = 300
with tqdm(desc="Corpus processed", total=max_validation_examples) as pbar_outer:
  for i, line in enumerate(validation_data):
      if i == max_validation_examples:
          break
      q, *ex = line
      ranks = rank_candidates(q, ex, embeddings_trained, tokenizer, dim=300)
      wv_ranking.append([r[0] for r in ranks].index(0) + 1)
      pbar_outer.update(1)
```

```
Corpus processed: 100%                                                1000/1000 [01:34<00:00, 12.03it/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_count(wv_ranking, k)))
```

```
100%                                           6/6 [00:00<00:00, 162.68it/s]
DCG@   1: 0.267 | Hits@   1: 0.267
DCG@   5: 0.342 | Hits@   5: 0.409
DCG@  10: 0.360 | Hits@  10: 0.466
DCG@ 100: 0.415 | Hits@ 100: 0.733
DCG@ 500: 0.439 | Hits@ 500: 0.922
DCG@1000: 0.447 | Hits@1000: 1.000
```

Видим, что качество модели, обученной на полученных эмбеддингах, довольно существенно уступает результатам ранжирования с использованием ранее обученного словаря wv_embeddings.

Попробуем более сложную кастомную токензацию с использвоанием библиотеки NLTK:

1) приведем все слова в корпусе к нижнему регистру;

2) удалим стоп-слова;

3) проведем лемматизацию

При этом знаки пунктуации и проч служебные символы удалять не будем.

```
import nltk
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
stopwords = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
class MyComplexTokenizer:
    def __init__(self):
        pass
    def tokenize(self, text):
      text = text.lower()
      lemmatizer = WordNetLemmatizer()
      tokenized = nltk.word_tokenize(text)
      filtered_sentence = [w for w in tokenized if not w in stopwords]
      lemmatized_output = ' '.join([lemmatizer.lemmatize(w) for w in filtered_sentence])
      text_only = re.sub(r"[^a-z0-9!@#\$%\^\&\*_\-,\.' ]", ' ', lemmatized_output)
      final = ' '.join(text_only.split())
      return final


tokenizer = MyComplexTokenizer()
```

```
wv_ranking = []
max_validation_examples = 1000
dim = 300
with tqdm(desc="Corpus processed", total=max_validation_examples) as pbar_outer:
  for i, line in enumerate(validation_data):
      if i == max_validation_examples:
          break
      q, *ex = line
      ranks = rank_candidates(q, ex, embeddings_trained, tokenizer, dim=300)
      wv_ranking.append([r[0] for r in ranks].index(0) + 1)
      pbar_outer.update(1)
```

```
Corpus processed: 100%                                                  1000/1000 [06:46<00:00,  2.38it/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, 150.96it/s]
DCG@   1: 0.106 | Hits@   1: 0.106
DCG@   5: 0.135 | Hits@   5: 0.162
DCG@  10: 0.148 | Hits@  10: 0.205
DCG@ 100: 0.197 | Hits@ 100: 0.453
DCG@ 500: 0.241 | Hits@ 500: 0.804
DCG@1000: 0.262 | Hits@1000: 1.000
```

Видим, что после указанной нормализации словаря качество метрик значительно ухудшилось. Предположим, что нормализация слов уменьшает количество значимой информации, необходимой для обучения эмбеддингов: видимо, важен и регистр и разные словоформы. В таких условиях логично:

1) увеличить размер окна;

2) при этом размер скрытого слоя снизить, дабы модель смогла усвоить как можно более значимые признаки;

3) с помощью модели Skip-Gram постараемся получить значимую инфу о контексте каждого токена (архитектура Skip-Gram лучше CBOW позволяет представлять редкие слова или фразы).

```
embeddings_trained = Word2Vec(proc_words, # data for model to train on
                 vector_size=200,                 # embedding vector size
                 min_count=9,             # consider words that occured at least 5 times
                 window=10,
                        sg=1).wv #


tokenizer = MyTokenizer()
```

```
wv_ranking = []
max_validation_examples = 1000
dim = 200
with tqdm(desc="Corpus processed", total=max_validation_examples) as pbar_outer:
  for i, line in enumerate(validation_data):
      if i == max_validation_examples:
          break
      q, *ex = line
      ranks = rank_candidates(q, ex, embeddings_trained, tokenizer, dim=dim)
      wv_ranking.append([r[0] for r in ranks].index(0) + 1)
      pbar_outer.update(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)))
```

```
100%                                                 6/6 [00:00<00:00, 173.12it/s]
DCG@   1: 0.432 | Hits@   1: 0.432
DCG@   5: 0.508 | Hits@   5: 0.574
DCG@  10: 0.529 | Hits@  10: 0.639
DCG@ 100: 0.570 | Hits@ 100: 0.839
DCG@ 500: 0.584 | Hits@ 500: 0.950
DCG@1000: 0.590 | Hits@1000: 1.000
```

Видим, что наши предположения оказались верны: получившиеся эмбеддинги даде чуть превосходят результаты ранжирования с использованием ранее обученного словаря wv_embeddings.

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

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:

1. Опытным путем выяснили, что лучшее качество для данных текстов дает довольной минимилистичный, кастомный способ токенизации, когда остаются только буквенные токены с сохранением их словоформ. Видимо, сказывается специфика текстов StackOverFlow: наполненность техническими словами и смыслами при маленькой длине предложений (до "склейки" < 10, тогда как из открытых источников ср длина предложения в русском языке ~ 20 токенов), широкое использование аббревиатур и обозначений, учитывающих регистр, и тп.
2. Исходя из сказанного выше, понятно, что нормализация слов будет только ухудшать качество.
3. После "склейки" пар слов в корпусе и исследования среднией и минимальной длин вопросов в получившемся корпусе был подобран диапазон для поиска оптимальных гиперпарамтеров, главным образом, размер окна. В результате перебора различных вариантов подобраны следующие гиперпараметры:
    - `vector_size` =200,
    - `min_count` =9,
    - `window` =10,
    - архитектура - Skip-Gram Обоснование, почему такие эмбеддинги могут работать лучше, приведено выше.
4. Качество решения, действительно, получилось не очень. При этом есть подозрения, что, как бы мы не старались, в рамках архитектуры word2vec мы не добьемся существенного увеличения качества. По-прежнему остаются недостатки:
    - Обучение на уровне слов: нет информации о предложении или контексте, в котором используется слово.
    - Совместная встречаемость игнорируется. Модель не учитывает то, что слово может иметь различное значение в зависимости от контекста использования.
    - Не очень хорошо обрабатывает неизвестные и редкие слова.
5. Наиболее очевидное решение использовать те модели, которые минимизируют указанные недостатки: так модель GloVe (Global Vectors) на базе того же word2vec учитывает частоту встречаемости слов и, как правило, опережает word2vec на большинстве бенчмарков. Недостаток с неизвестными и редкими словами может "закрыть" fastText, которая к основной модели word2vec добавлет модель символьных n-грамм (способна генерировать эмбеддинги и для неизвестных слов).