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

✓ 0 сек. выполнено в 17:16

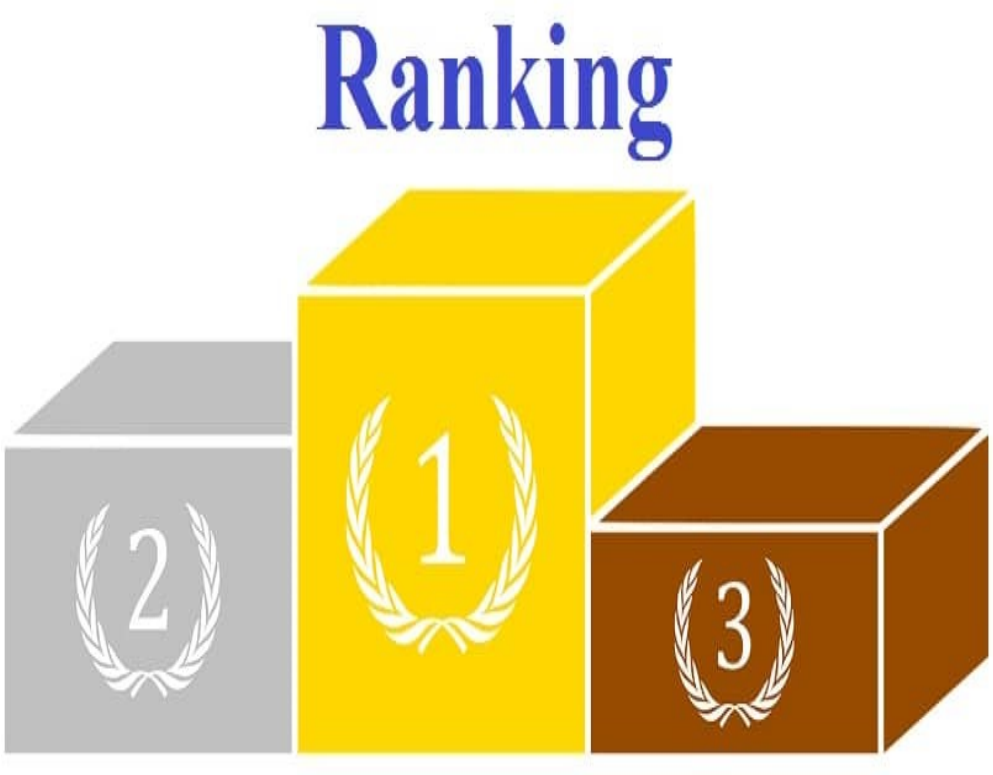
Задача ранжирования(Learning to Rank)

- 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]

```
# !pip install gensim nltk spacy
```

```
# https://zenodo.org/record/1199620/files/SO_vectors_200.bin?download=1
from gensim.models.keyedvectors import KeyedVectors
from numpy.linalg import norm
import numpy as np
import re
import nltk
from nltk import WordPunctTokenizer
from nltk import SpaceTokenizer
import pandas as pd
from functools import cmp_to_key
from tqdm.notebook import tqdm
from gensim.models import Word2Vec
from sklearn.metrics.pairwise import cosine_similarity
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, SnowballStemmer
import re
import string
from nltk.stem import WordNetLemmatizer
```

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

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

```
# import ssl

# try:
#     _create_unverified_https_context = ssl._create_unverified_context
# except AttributeError:
#     pass
# else:
#     ssl._create_default_https_context = _create_unverified_https_context

# nltk.download('stopwords')
# nltk.download('wordnet')
```

```
wv_embeddings = KeyedVectors.load_word2vec_format("/content/drive/MyDrive/PH2Dataset/SO_
```

```
# wv_embeddings.index_to_key
```

Как пользоваться этими векторами?

Посмотрим на примере одного слова, что из себя представляет embedding

```
word = 'dog'
```

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

```
float32 (200,)
```

```
cosine_similarity([wv_embeddings['dog']], [wv_embeddings['dog']])
```

```
array([[1.0000001]], dtype=float32)
```

```
print(f"Num of words: {len(wv_embeddings.index_to_key)}")
```

```
Num of words: 1787145
```

```
dog_v = wv_embeddings[word]
```

Найдем наиболее близкие слова к слову `dog`:

Вопрос 1:

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

Ответ 1: Кот не в топ 5. Место 25.

```
# С помощью встроенной функции
def get_most_similar_words_gensim(embeddings, word, n_words):
    return embeddings.most_similar(positive=[word], topn=n_words)
```

```
top5_sim_dog = get_most_similar_words_gensim(wv_embeddings, 'dog', 5)
top100_sim_dog = get_most_similar_words_gensim(wv_embeddings, 'dog', 100)
cat_pos_top5 = [index for (index, x) in enumerate(top5_sim_dog) if x[0] == 'cat']
if cat_pos_top5 :
    cat_pos_top5[0]
else:
    print('Кот не в топ 5')

cat_pos_top100 = [index for (index, x) in enumerate(top100_sim_dog) if x[0] == 'cat']
if cat_pos_top100:
    print(f'Позиция кота {cat_pos_top100[0]}')
```

```
Кот не в топ 5
Позиция кота 25
```

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.

```
# you can use your tokenizer
# for example, from nltk.tokenize import WordPunctTokenizer
class MyTokenizer:
    def __init__(self):
        self.tokenizer = WordPunctTokenizer()
    def tokenize(self, text):
        return self.tokenizer.tokenize(text)
```

```
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
    """
    tokens = tokenizer.tokenize(question)
    vectors = [embeddings[x] for x in tokens if x in embeddings]

    if len(vectors) == 0 :
        return np.zeros(dim)

    stacked_v = np.stack(vectors)

    return np.mean(stacked_v, axis=0)
```

Вопрос 2:

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

Ответ 2: -1.29

```
question = 'I love neural networks'
tokenizer = MyTokenizer()

round(question_to_vec(question, wv_embeddings, 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::

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

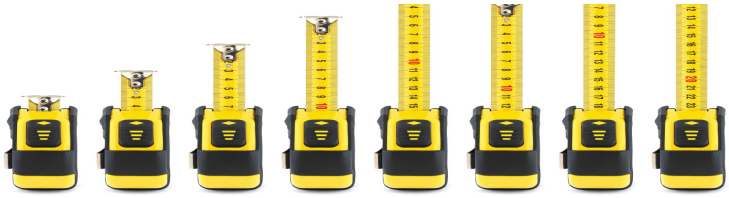
С такой метрикой модель штрафуется за большой ранк корректного ответа

Вопрос 3:

- Максимум `Hits@47 - DCG@1` ?

Ответ 3: max(HITS@47 - DCG@1) =max(HITS@47) - min(DCG@1) = 1

- max(DCG@1) = 1
- max(Hits@47) = 1
- min(DCG@1) = 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) (N = 1 ?)

Ответ 4: 0.3

```
round(1 / np.log2(10), 1)
```

```
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
    """
    return sum([1 if x <= k else 0 for x in dup_ranks]) / len(dup_ranks)


def dcg_score(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        result: вернуть DCG@k
    """
    return sum([1 / np.log2(x + 1) if x <= k  else 0 for x in dup_ranks]) / len(dup_rank
```

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

```
copy_answers = ["How does the catch keyword determine the type of exception that was thr

# наги кандидаты
candidates_ranking = [["How Can I Make These Links Rotate in PHP",
                       "How does the catch keyword determine the type of exception that
                       "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
                                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

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>, ...**

Считайте данные.

```
def read_corpus(filename, val=False):
    data = []
    for line in open(filename, encoding='utf-8'):
        splitted = line.split('\t')
        data.append((splitted[0], splitted[1:]))
    return data
```

Нам понадобиться только файл validation.

```
validation_data = read_corpus('/content/drive/MyDrive/PH2Dataset/data/validation.tsv')
```

```
train_data = read_corpus('/content/drive/MyDrive/PH2Dataset/data/train.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 2
2 2
3 2
4 2
5 2
```

```
q, ex = validation_data[0]
```

```
[x for x in train_data if x[0] == 'Sending array via Ajax fails']
```

```
[('Sending array via Ajax fails',
  ['Getting all list items of an unordered list in PHP\n'])]
```

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)]**.

```
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, dim)
    candidates_vecs = [(index, question_to_vec(x, embeddings, tokenizer, dim)) for (inde
    return sorted(enumerate(candidates), key=lambda x: -cosine_similarity([candidates_ve
```

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 эксп
               '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 экспе
               '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)
```

```
[(1, 'C# create cookie from string and send it'), (0, 'Convert Google results obje
[(1, 'WPF- How to update the changes in list item of a list'), (0, 'Getting all li
```

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?')],
#             [(1, 'WPF- How to update the changes in list item of a list'), #скрыт
#              (0, 'Getting all list items of an unordered list in PHP'), #скрыт
#              (2, 'select2 not displaying search results') #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.

```
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 [05:31<13:31, 3.40it/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
```

```
  100%                                                 6/6 [00:00<00:00, 203.33it/s]
DCG@   1: 0.276 | Hits@   1: 0.276
```

```
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
```

Эмбеддинги, обученные на корпусе похожих вопросов

```
train_data = read_corpus('/content/drive/MyDrive/PH2Dataset/data/train.tsv')
```

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

```
class NoStopsPuncTokenizer:
    def __init__(self, lang = 'english'):
        self.tokenizer = WordPunctTokenizer()
        self.lang = lang
        self.stops = set(stopwords.words(self.lang))

    def tokenize(self, text):
        return [x for x in self.tokenizer.tokenize(text) if x not in self.stops]

stops_filter_tokenizer_punct = NoStopsPuncTokenizer()
```

```
class NoStopsPuncStemmaTokenizer:
    def __init__(self, lang = 'english'):
        self.tokenizer = WordPunctTokenizer()
        self.lang = lang
        self.ps = PorterStemmer()
        self.stops = set(stopwords.words(self.lang))

    def tokenize(self, text):
        return [self.ps.stem(x) for x in self.tokenizer.tokenize(text) if x not in self.
```

```
no_stops_stemma_tonekizer = NoStopsPuncStemmaTokenizer()
```

```
class NoStopsNoPuncTokenizer:
    def __init__(self, lang = 'english'):
        self.tokenizer = WordPunctTokenizer()
        self.lang = lang
        self.stops = set(stopwords.words(self.lang))

    def tokenize(self, text):
        return [x for x in self.tokenizer.tokenize(text) if x not in self.stops and x nc
```

```
stops_filter_tokenizer_no_punct = NoStopsNoPuncTokenizer()
```

```
class NoStopsNoPuncLemmaTokenizer:
    def __init__(self, lang = 'english'):
        self.tokenizer = WordPunctTokenizer()
        self.lang = lang
        self.stops = set(stopwords.words(self.lang))
        self.lemmatizer = WordNetLemmatizer()

    def tokenize(self, text):
        return [self.lemmatizer.lemmatize(x).lower() for x in self.tokenizer.tokenize(te

no_stops_no_punct_lemma_tokenizer = NoStopsNoPuncLemmaTokenizer()


window = 10 # В окно 10 в большинстве случаев будут попадать слова как из начальаного вс


sentences = [x[0] + ' ' +  x[1][0] for x in train_data]


no_stops_no_punct_lemma_tokenizer.tokenize(sentences[0])
```

```
['converting',
 'string',
 'list',
 'convert',
 'google',
 'result',
 'object',
 'pure',
 'j',
 'python',
 'object']
```

```
words = [no_stops_stemma_tonekizer.tokenize(x) for x in tqdm(sentences)]
words_len = 0
for x in words:
    words_len+= len(x)
print(words_len)
```

```
100% 1000000/1000000 [04:11<00:00, 4505.53it/s]
14884932
```

```
no_stops_words_punct = [stops_filter_tokenizer_punct.tokenize(x) for x in tqdm(sentences
no_stops_words_len = 0
for x in no_stops_words_punct:
    no_stops_words_len+= len(x)
print(no_stops_words_len)
```

```
100% 1000000/1000000 [00:16<00:00, 57696.14it/s]
14884932
```

```
no_stops_no_punct_words = [stops_filter_tokenizer_no_punct.tokenize(x) for x in tqdm(ser
```

```
l2 = 0
for x in no_stops_no_punct_words:
    l2 += len(x)
print(l2)
```

```
100%                                                1000000/1000000 [00:19<00:00, 74604.41it/s]
12675976
```

```
no_stops_no_punct_lemma_words = [no_stops_no_punct_lemma_tokenizer.tokenize(x) for x in
l3 = 0
for x in no_stops_no_punct_lemma_words:
    l3 += len(x)
print(l3)
```

```
100%                                                1000000/1000000 [01:08<00:00, 17174.46it/s]
11793814
```

```
embeddings_trained = Word2Vec(no_stops_no_punct_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=15).wv
```

```
govno2 = (embeddings_trained['list'] + embeddings_trained['array'])/2
cosine_similarity([govno2], [embeddings_trained['list']]), cosine_similarity([govno2], [
```

```
(array([[0.8477517]], dtype=float32), array([[0.88486904]], dtype=float32))
```

```
top200 = get_most_similar_words_gensim(embeddings_trained, 'list', 300000)
s_pos_top200 = [index for (index, x) in enumerate(top200) if x[0] == 'array']
if s_pos_top200:
    print(f'Позиция array {s_pos_top200[0]}')
```

```
Позиция array 5
```

```
get_most_similar_words_gensim(embeddings_trained, 'list', 5)
```

```
[('lists', 0.663406252861023),
 ('sublist', 0.6008540987968445),
 ('List', 0.6008093357086182),
 ('tuple', 0.5420591235160828),
 ('arraylist', 0.5161855220794678)]
```

```
print(f"Num of words: {len(embeddings_trained.index_to_key)}")
```

```
Num of words: 49312
```

```
wv_ranking = []
```

```
max_validation_examples = 2000
for i, line in enumerate(tqdm(validation_data)):
    if i == max_validation_examples:
        break
    q, *ex = line
    ex = ex[0]
    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
```

```
 53%                                                  2000/3760 [11:07<08:48, 3.33it/s]

 100%                                                  6/6 [00:00<00:00, 114.29it/s]
```

```
DCG@   1: 0.348 | Hits@   1: 0.348
DCG@   5: 0.441 | Hits@   5: 0.521
DCG@  10: 0.469 | Hits@  10: 0.606
DCG@ 100: 0.516 | Hits@ 100: 0.837
DCG@ 500: 0.532 | Hits@ 500: 0.961
DCG@1000: 0.536 | Hits@1000: 1.000
```

Лучший результат: min_count=5, window=15, WordPunctTokenizer с фильтрацией стоп слов и лемматизацией Результаты:

```
DCG@   1: 0.439 | Hits@   1: 0.439
DCG@   5: 0.530 | Hits@   5: 0.609
DCG@  10: 0.554 | Hits@  10: 0.684
DCG@ 100: 0.595 | Hits@ 100: 0.881
DCG@ 500: 0.608 | Hits@ 500: 0.977
DCG@1000: 0.610 | 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:

Вывод:

1. Токенизация по словам с пунктуацией, с удалением стоп слов. Токенизация больше чем по одну слову на токен не рассматривалась, так как работаем с предложениеми. С фильтром стоп слов качество лучше, для векторных представлений вопросов они только мешали.

2. Да, стемминг и лемматизация показывает лучшие результаты.

3. Обучать с помощью Word2Vec на вопросах из train лучше, потому что можно сильнее учитывать контекст в построении эмбеддингов.

4. Проблема в том, как мы составляем эмбедденги предложений. Среднее по эмбеддингам слов - не лучший способ.

5. По-другому составлять эмбедденги предложений. Взять RNN модель, выкинуть голову, и прогонять вопросы через получившуюся модель. Новые эмбедденги ранжировать с помощью косинусного расстояния.

Bonus track: Grid Search

```
def tokenize_sentences(sentences, tokenizer):
    tokenized_sentences = [tokenizer.tokenize(x) for x in tqdm(sentences)]
    l = 0
    for x in tokenized_sentences:
        l += len(x)
    print(f'Всего токенов получилось: {l}')
    return tokenized_sentences

def train_embeddings(tokenized_sentences, min_count, window):
    embeddings_trained = Word2Vec(tokenized_sentences,
                                  vector_size=200,
                                  min_count=min_count,
                                  window=window).wv
    return embeddings_trained


def get_results(validation_data, embeddings_trained, tokenizer, max_val_examples):
    wv_ranking = []
    max_validation_examples = max_val_examples
    for i, line in enumerate(validation_data):
        if i == max_validation_examples:
            break
```

```
        q, *ex = line
        ex = ex[0]
        ranks = rank_candidates(q, ex, embeddings_trained, tokenizer)
        wv_ranking.append([r[0] for r in ranks].index(0) + 1)

    results = []
    for k in [1, 5, 10, 100, 500, 1000]:
        dcg = dcg_score(wv_ranking, k)
        hits = hits_count(wv_ranking, k)
        results.append((k, round(dcg,3), round(hits, 3)))
        print("DCG@%4d: %.3f | Hits@%4d: %.3f" % (k, dcg, k, hits))

    return results


def word2VecGridSearch(sentences, val_data,  min_counts, windows, tokenizers, max_val_ex
    """
    :return: the best combination of parameters with results
    parameters format ([min_count, window, tokenizer])
    results format [[k], [dcg], [hits]]
    """
    results = []
    for tokenizer in tokenizers:
        tokenized_sentences = tokenize_sentences(sentences, tokenizer)
        for min_count in min_counts:
            for window in windows:
                print()
                print(f'mc = {min_count}, w = {window} t = {tokenizer.__class__}')
                print('------------------------------------------------------------')
                embeddings_trained = train_embeddings(tokenized_sentences, min_count, wi
                results.append(((min_count, window, tokenizer.__class__),get_results(val
                print()

    best = max(results, key=lambda x: x[1][0][1])
    print(f'Лучший: mc = {best[0][0]}, w = {best[0][1]}, tokeizer={best[0][2]}')
    print(f'Результаты: {best[1]}')

    return results


grid_search_result = word2VecGridSearch(sentences,
                  validation_data,
                  [5],
                  [15],
                  [stops_filter_tokenizer_no_punct, no_stops_stemma_tonekizer, no_stops
                  2000)
```

```
 100%                                                1000000/1000000 [00:22<00:00, 3766.01it/s]
 Всего токенов получилось: 12675976

 mc = 5, w = 15 t = <class '__main__.NoStopsNoPuncTokenizer'>
 ------------------------------------------------------------
 DCG@   1: 0.348 | Hits@   1: 0.348
 DCG@   5: 0.441 | Hits@   5: 0.523
 DCG@  10: 0.468 | Hits@  10: 0.607
```

```
DCG@  10: 0.468 | Hits@  10: 0.607
DCG@ 100: 0.516 | Hits@ 100: 0.837
DCG@ 500: 0.532 | Hits@ 500: 0.963
DCG@1000: 0.536 | Hits@1000: 1.000
```

```
100%                                                1000000/1000000 [05:15<00:00, 3708.39it/s]
```

```
Всего токенов получилось: 14884932

mc = 5, w = 15 t = <class '__main__.NoStopsPuncStemmaTokenizer'>
-----------------------------------------------------------
DCG@   1: 0.411 | Hits@   1: 0.411
DCG@   5: 0.499 | Hits@   5: 0.577
DCG@  10: 0.521 | Hits@  10: 0.643
DCG@ 100: 0.567 | Hits@ 100: 0.862
DCG@ 500: 0.581 | Hits@ 500: 0.972
DCG@1000: 0.584 | Hits@1000: 1.000
```

```
72%                                                 724258/1000000 [00:54<00:18, 15217.33it/s]
```

```
Всего токенов получилось: 11793814

mc = 5, w = 15 t = <class '__main__.NoStopsNoPuncLemmaTokenizer'>
-----------------------------------------------------------
DCG@   1: 0.439 | Hits@   1: 0.439
DCG@   5: 0.530 | Hits@   5: 0.609
DCG@  10: 0.554 | Hits@  10: 0.684
DCG@ 100: 0.595 | Hits@ 100: 0.881
DCG@ 500: 0.608 | Hits@ 500: 0.977
DCG@1000: 0.610 | Hits@1000: 1.000

Лучший: mc = 5, w = 15, tokeizer=<class '__main__.NoStopsNoPuncLemmaTokenizer'>
Результаты: [(1, 0.438, 0.439), (5, 0.53, 0.609), (10, 0.554, 0.684), (100, 0.595,
```

Лучший: mc = 5, w = 15, tokeizer=<class '**main**.NoStopsNoPuncLemmaTokenizer'>
Результаты: [(1, 0.438, 0.439), (5, 0.53, 0.609), (10, 0.554, 0.684), (100, 0.595, 0.881), (500, 0.608, 0.977), (1000, 0.61, 1.0)]

Платные продукты Colab - Отменить подписку