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

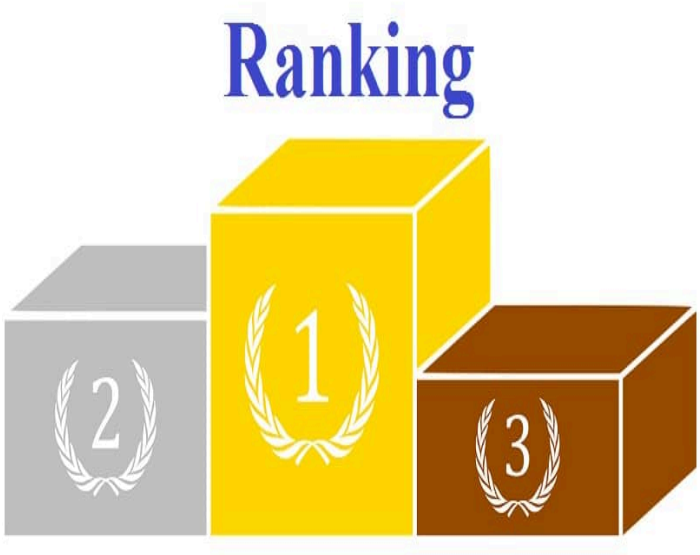
Задача ранжирования(Learning to Rank)

- 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
```

```
--2024-03-04 18:55:17--  https://zenodo.org/record/1199620/files/SO_vectors_200.bin?download=1
Resolving zenodo.org (zenodo.org)... 188.184.103.159, 188.184.98.238, 188.185.79.172, ...
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]
--2024-03-04 18:55:18--  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  23.3MB/s    in 61s

2024-03-04 18:56:20 (22.6 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?

```
five_similar_words = wv_embeddings.most_similar("dog", topn=5)
print(five_similar_words)

cat_rank = None
for rank, (word, similarity) in enumerate(five_similar_words, 1):
    if word == "cat":
        cat_rank = rank - 1
        break
print(f"Rank of the word 'cat' out of top-5: {cat_rank}")


cats_rank = None
for place, (word, similarity) in enumerate(five_similar_words, 1):
    if word == "cats":
        cats_rank = rank - 1
        break
print(f"Rank of the word 'cats' out of top-5: {cats_rank}")
```

```
[('animal', 0.8564180135726929), ('dogs', 0.7880866527557373), ('mammal', 0.7623804211616516), ('cats', 0.7621253
Rank of the word 'cat' out of top-5: None
Rank of the word 'cats' out of top-5: 4
```

Ответ: Мы видим, что слово 'cat' не входит в число 5 наиболее близких слов к слову 'dog'. Однако: в этом топе присутсвует множественное число - 'cats' (4 место). Значит, нам нужно научиться определять близость вне зависимости от числа.

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
from nltk.tokenize import WordPunctTokenizer
class MyTokenizer(object):
    def __init__(self):
        pass
    def tokenize(self, text):
        return re.findall('\w+', 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
    """

    words = tokenizer.tokenize(question)  # Tokenizing
    vec_sum = np.zeros(dim)
    count = 0

    for word in words:
        if word in embeddings:
            vec_sum += embeddings[word]
            count += 1

    if count > 0:
        return vec_sum / count
    else:
        return np.zeros(dim)
```

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)?

```
tokenizer = MyTokenizer()
sentence_vector = question_to_vec("I love neural networks", wv_embeddings, tokenizer)
print(f"{sentence_vector[2]:.2f}")
```

```
-1.29
```

Ответ: -1.29

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

class SpacyTokenizer:
    def __init__(self):
        pass
    def tokenize(self, text):
        spacy_results = nlp(text)
        return [token.lemma_ for token in spacy_results]

spacy_tokenizer = SpacyTokenizer()
sentence_vector = question_to_vec("I love neural networks", wv_embeddings, tokenizer)
print(f"{sentence_vector[2]:.2f}")
```

```
-1.29
```

```
tokenizer = WordPunctTokenizer()
sentence_vector = question_to_vec("I love neural networks", wv_embeddings, tokenizer)
print(f"{sentence_vector[2]:.2f}")
```

```
-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`?

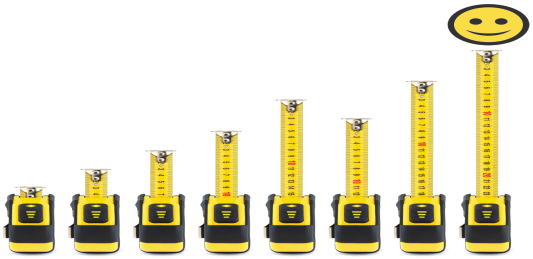
$$\max\left(\frac{1}{N}\sum_{i=1}^{N}(rank\_q_i' \le 47) - \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\log_2(1+rank\_q_i')}\cdot[rank\_q_i' \le 1]\right) =$$

$$= 1 - 0 = 1$$

Максимальной разница будет, если одновременно Hits@47 будем максимальна, а DCG@1 будет минимальна. Такое возможно, например: N = 1, rank = 2

Ответ: 1

```
# Выбран кодовый формат
```



Пример оценок

Вычислим описанные выше метрики для игрушечного примера. Пусть

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

Пусть модель выдала следующий ранжированный список кандидатов:

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

$\Rightarrow rank\_q_i^{'} = 2$

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

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

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

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

Вопрос 4:

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

$$\text{DCG@10} = \frac{1}{\log_2(1+9)} = \frac{1}{\log_2(10)} \approx \frac{1}{3.3219} \approx 0.3$$

Ответ: 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 = sum([1 for rank in dup_ranks if rank <= k]) / len(dup_ranks)
    return hits_value


import math
def dcg_score(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        result: вернуть DCG@k
    """
    dcg_value = round(sum([1 / math.log2(1 + rank) if rank <= k else 0 for rank in dup_ranks]) / len(dup_ranks),5)
    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[0].index(copy_answers[0]) + 1]

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

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 stackoverflow_similar_questions.zip
```

```
Archive:  stackoverflow_similar_questions.zip
   creating: data/
```

```
        inflating: data/.DS_Store
         creating: __MACOSX/
         creating: __MACOSX/data/
        inflating: __MACOSX/data/._.DS_Store
        inflating: data/train.tsv
        inflating: data/validation.tsv
```

Считайте данные.

```
def read_corpus(filename):
    data = []
    for line in open(filename, encoding='utf-8'):
        data.append(line.strip())
    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 54516
2 52817
3 54463
4 53723
5 52296
```

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):
    """
    Rank the candidates based on their similarity to the question using word embeddings.

    Args:
        question: Input question string
        candidates: List of candidate strings [a, b, c]
        embeddings: Word embeddings model
        tokenizer: Tokenizer for processing text
        dim: Dimension of the word embeddings (default is 200)

    Returns:
        List of ranked candidates in the format [(initial position, candidate), ...]
    """
    question_vec = question_to_vec(question, embeddings, tokenizer, dim)
    candidate_vecs = [question_to_vec(candidate, embeddings, tokenizer, dim) for candidate in candidates]

    similarity_scores = []
    for candidate_vec in candidate_vecs:
        norm_question = np.linalg.norm(question_vec)
        norm_candidate = np.linalg.norm(candidate_vec)
        if norm_question == 0 or np.isnan(norm_question) or norm_candidate == 0 or np.isnan(norm_candidate):
            similarity_scores.append(0)
        else:
            similarity_scores.append(np.dot(question_vec, candidate_vec) / (norm_question * norm_candidate))

    ranked_candidates = sorted(list(enumerate(candidates)), key=lambda x: similarity_scores[x[0]], reverse=True)
    return ranked_candidates
```

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 i
```

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: 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
```

```
from IPython.display import clear_output
tokenizer = MyTokenizer()

wv_ranking = []
max_validation_examples = 1000
for i, line in enumerate(tqdm(validation_data)):
    print(i)
    clear_output(wait=True)
    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)
```

```
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, 184.92it/s]
DCG@   1: 0.904 | Hits@   1: 0.904
DCG@   5: 0.904 | Hits@   5: 0.904
DCG@  10: 0.904 | Hits@  10: 0.904
DCG@ 100: 0.904 | Hits@ 100: 0.904
DCG@ 500: 0.904 | Hits@ 500: 0.907
DCG@1000: 0.904 | Hits@1000: 0.907
```

```
tokenizer = WordPunctTokenizer()

WordPunct_ranking = []
max_validation_examples = 1000
for i, line in enumerate(tqdm(validation_data)):
    print(i)
    clear_output(wait=True)
    if i == max_validation_examples:
        break
    q, *ex = line
    ranks = rank_candidates(q, ex, wv_embeddings, tokenizer)
    WordPunct_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(WordPunct_ranking, k), k, hits_count(WordPunct_ranking, k)
```

```
100%                                          6/6 [00:00<00:00, 161.64it/s]
DCG@   1: 0.903 | Hits@   1: 0.903
DCG@   5: 0.903 | Hits@   5: 0.903
DCG@  10: 0.903 | Hits@  10: 0.903
DCG@ 100: 0.903 | Hits@ 100: 0.903
DCG@ 500: 0.904 | Hits@ 500: 0.907
DCG@1000: 0.904 | Hits@1000: 0.907
```

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.

Window = 5, тк это значение одновременно улавливаниет локальную информацию в предложении, однако несет в себе и более серьёзную смысловую нагрузку.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

nltk.download('punkt')
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 stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
from gensim.models import Word2Vec

# Concatenate question pairs
question_pairs = ["How to concatenate strings?", "What is the best way to split a string?"]
concatenated_data = [q1 + " " + q2 for q1, q2 in zip(question_pairs[::2], question_pairs[1::2])]


def preproc_nltk(text): # adding this as in seminar
    return ' '.join([word for word in word_tokenize(text.lower()) if word not in stopWords])

couple_list = [couple[0] + ' ' + couple[1] for couple in train_data]
words = [preproc_nltk(question).split() for question in couple_list]
embedding_models = []

for sg_value in [1, 0]:
    embeddings_trained = Word2Vec(words,
                                  vector_size=200,
                                  min_count=5,
                                  window=5,
                                  workers=4,
                                  sg=sg_value).wv
    wv_ranking = []
    max_validation_examples = 1000
    tokenizer = MyTokenizer()

    for i, line in enumerate(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)

    model_type = "Skip-Gram" if sg_value == 1 else "CBOW"
    print(f'Model Type: {model_type}')

    for k in [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)))

    embedding_models.append(embeddings_trained)
```

```
Model Type: Skip-Gram
DCG@   1: 0.904 | Hits@   1: 0.904
DCG@   5: 0.904 | Hits@   5: 0.904
DCG@  10: 0.904 | Hits@  10: 0.904
DCG@ 100: 0.904 | Hits@ 100: 0.905
DCG@ 500: 0.904 | Hits@ 500: 0.906
DCG@1000: 0.904 | Hits@1000: 0.906
Model Type: CBOW
DCG@   1: 0.904 | Hits@   1: 0.904
DCG@   5: 0.904 | Hits@   5: 0.904
DCG@  10: 0.904 | Hits@  10: 0.904
DCG@ 100: 0.904 | Hits@ 100: 0.905
DCG@ 500: 0.904 | Hits@ 500: 0.906
DCG@1000: 0.904 | Hits@1000: 0.906
```

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:

На основании полученных результатов:

1. **Какой принцип токенизации даёт качество лучше и почему?**:
    - MyTokenize() немного превосходит WordPunctTokenize(). Основное различие между этими двумя токенизаторами заключается в регулярных выражениях, которые они используют для токенизации текста:
    - WordPunctTokenizer разделяет текст на слова и знаки препинания, сохраняя слова нетронутыми. MyTokenizer захватывает последовательности словесных символов, эффективно разделяя текст на слова.
    - Что касается того, почему MyTokenizer может считаться лучше, чем WordPunctTokenizer, то это зависит от конкретных требований поставленной задачи. Вот некоторые соображения: