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

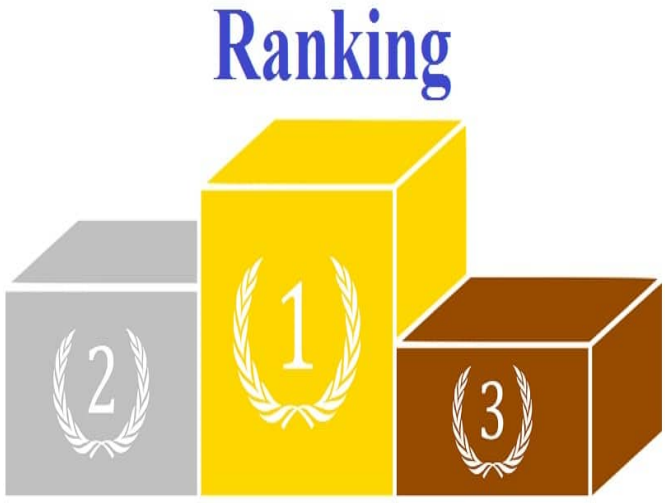
Задача ранжирования(Learning to Rank)

- 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
```

```
--2023-03-13 03:09:48--  https://zenodo.org/record/1199620/files/SO_vectors_200.bin?download=1
Resolving zenodo.org (zenodo.org)... 188.185.124.72
Connecting to zenodo.org (zenodo.org)|188.185.124.72|:443... connected.
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  9.07MB/s    in 2m 58s

2023-03-13 03:12:48 (7.80 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.index2word)}")
```

```
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
'''your code'''
print(wv_embeddings.most_similar(positive=[word, 'cat'])[:5])
```

```
[('feline', 0.7719643115997314), ('cats', 0.7606382369995117), ('animal', 0.7583404779434204), ('meow', 0.7338752746582(
```

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
    """

    '''your code'''
    ret = np.zeros((200,), dtype=float)
    count = 0
    for item in tokenizer.tokenize(question):
      # if embeddings.get(item).shape[0] > 0:
      try:
        # print(embeddings[item].shape)
        ret += embeddings[item]
        count += 1
      except:
        ...
    if count != 0:
      ret /= count
    return ret
# question_to_vec('dog barks cat you', wv_embeddings, tokenizer)
```

```
# wv_embeddings[word]
```

Теперь у нас есть метод для создания векторного представления любого предложения.

Вопрос 2:

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

```
'''your code'''
round(question_to_vec('I love neural networks', 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

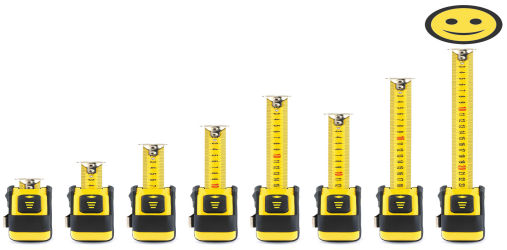
Второй метрикой будет упрощенная DCG метрика, учитывающая порядок элементов в списке путем домножения релевантности элемента на вес равный обратному логарифму номера позиции::

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

С такой метрикой модель штрафуется за большой ранк корректного ответа

Вопрос 3:

- Максимум `Hits@47 - DCG@1`?



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)

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
    """
```

```
    '''your code'''
    hits_value = sum(np.array(dup_ranks) <= k)
    return hits_value


def dcg_score(dup_ranks, k):
    """
        dup_ranks: list индексов дубликатов
        result: вернуть DCG@k
    """
    '''your code'''
    import math
    dcg_value = sum(list(map(lambda rating: (rating <= k) / math.log2(1+rating), dup_ranks)))
    return dcg_value
```

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

```
import pandas as pd


# candidates_ranking[0]


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
from sklearn.metrics.pairwise import cosine_similarity
vec1 = question_to_vec(copy_answers[0], wv_embeddings, tokenizer)
for count, item in enumerate(candidates_ranking[0]):
  vec2 = question_to_vec(item, wv_embeddings, tokenizer)
  # print(vec1.shape, vec2.shape)
  if abs(abs(cosine_similarity([vec1], [vec2])) - 1) < 1e-6:
    dup_ranks = [count+1]
    # print(count)
    break

# 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, 1, 1, 1]
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>, ...**

```
!gdown "1QqT4D0EoqJTy7v9VrNCYD-m964XZFR7_&confirm=t"
```

```
Downloading...
From: https://drive.google.com/uc?id=1QqT4D0EoqJTy7v9VrNCYD-m964XZFR7_&confirm=t
To: /content/stackoverflow_similar_questions.zip
100% 131M/131M [00:01<00:00, 108MB/s]
```

```
!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'):
        '''your code'''
        data.append(line)
    return data
```

Нам понадобиться только файл validation.

```
validation_data = read_corpus('/content/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 54517
2 52818
3 54464
4 53724
5 52297
```

```
validation_data[0]
```

```
'How to print a binary heap tree without recursion?\tHow do you best conve
rt a recursive function to an iterative one?\tHow can i use ng-model with
directive in angular js\tflash: drawing and erasing\ttoggle react componen
t using hide show classname\tUse a usercontrol from another project to cur
rent webpage\t~ Paths resolved differently after upgrading to ASP.NET 4\tM
aterialize datepicker - Rendering when an icon is clicked\tCreating PyPi p
```

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)]
    """
    '''your code'''
    ret = []
    vec1 = question_to_vec(question, embeddings, tokenizer)
    for count, item in enumerate(candidates):
      vec2 = question_to_vec(item, embeddings, tokenizer)
      ret.append((count, cosine_similarity([vec1], [vec2])[0][0], item))
```

```
    ret.sort(key=lambda x: x[1], reverse=True)
    return [(x[0], x[2]) for x in ret]
```

Протестируйте работу функции на примерах ниже. Пусть $N = 2$, то есть два эксперимента

```
# r = rank_candidates(questions[0], candidates[0], wv_embeddings, tokenizer)
# r.sort(key=lambda x: x[1], reverse=True)
# [(x[0], x[2]) for x in r]


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'), (2, '

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

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?')],
#           #  [(*, 'Getting all list items of an unordered list in PHP'), #скрыт
#           #   (*, 'select2 not displaying search results'), #скрыт
#           #   (*, '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?

```
# 0, 2, 1
```

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)
    try:
      wv_ranking.append([r[0] for r in ranks].index('0') + 1)
    except:
      ...
```

```
 27%                 1000/3760 [4:35:43<12:32:27, 16.36s/it]
```

```
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)))
```

Эмбеддинги, обученные на корпусе похожих вопросов

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

Улучшите качество модели.

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

```
'''your code'''
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('omw-1.4')
stopWords = set(stopwords.words('english'))
nltk.download('wordnet')
wnl = nltk.WordNetLemmatizer()


def preproc_nltk(text):
    #text = re.sub(f'[{string.punctuation}]', ' ', text)
    return ' '.join([wnl.lemmatize(word) for word in word_tokenize(text.lower()) if word not in stopWords])

words = [preproc_nltk(text).split() for text in train_data]
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
from gensim.models import Word2Vec
embeddings_trained = Word2Vec(words, # data for model to train on
                  size=200,                # embedding vector size
                  min_count=3, #'''your code''',            # consider words that occured at least 5 times
                  window=3).wv #window='''your code'''.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)
    try:
      wv_ranking.append([r[0] for r in ranks].index('0') + 1)
    except:
      ...


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:

- Были испробованы несколько подходов токенизации: выделение слов посредством регулярных выражений, регулярки + приведение к нижнему регистру + отбрасывание stopwords, регулярки + нижний регистр + лемматизация + отбрасывание stopwords, регулярки + нижний регистр + стемминг + отбрасывание stopwords. Наилучший результат показал подход "регулярки + приведение к нижнему регистру + отбрасывание stopwords"; лемматизация занимает слишком много времени.
- В проведенных экспериментах нормализация наоборот ухудшило качество модели(можно, наверное, списать на название домашнего задания "простой эмбеддинг", которое вводит в курс дела). В теории, нормализация должна повысить качество модели.
- В задании было рассмотрено 2 эмбеддинга: предобученный на корпусе вопросов со stackoverflow и обученный нами. Лучшее качество показал предобученный, это, скорее всего, связано с размером корпуса для этого эмбеддинга. Объема нашей тренировочной выборки по-видимому не хватает для обеспечения такого же качества.
- В данном задании были использованы простейшие методы для ранжирования схожих вопросов, поэтому не получилось достичь хорошего качества. Так же следовало бы изучить состав тренировочной выборки и тестовой: в тестовой выборки могут быть слова не встречающиеся в тренировочной, а векторное преставление нашего предложения находится как среднее по всем векторам, а новые слова задаются нулевым вектором. Такой подход и приводит к невысокому качеству решения задачи.
- Регулярки + приведение к нижнему регистру(возможно за исключением общепринятых аббревиатур) + отбрасывание stopwords + лемматизация + предобученный на большом корпусе вопросов stackoverflow эмбеддинг + другое представление предложения(возможно суммирование всех слов с весами + альтернативная обработка не встречавшихся слов)