Text Summarization. Homework

Всем привет! Это домашка по суммаризации текста.

На семинаре мы рассмотрели базовые модели для суммаризации текста. Попробуйте теперь улучшить два метода: TextRank и Extractive RNN. Задание достаточно большое и требует хорошую фантазию, тут можно эксперементировать во всю.

Для сдачи заданий надо получить определенное качество по test-у:

- 1 задание: 0.35 BLEU
- 2 задание: 0.35 BLEU

Если ваш подход пробивает это качество – задание считается пройденным. Плюсом будет описание того, почему вы решили использовать то или иное решение.

Датасет: gazeta.ru

P.S. Возможно, в датасете находятся пустые данные. Проверьте эту гипотезу, и если надо, сделайте предобратоку датасета.

Ноутбук создан на основе семинара Гусева Ильи на кафедре компьютерной лингвистики МФТИ.

Загрузим датасет и необходимые библиотеки

```
!wget -q https://www.dropbox.com/s/43l702z5a5i2w8j/gazeta_train.txt
!wget -q https://www.dropbox.com/s/k2egt3sug0hb185/gazeta_val.txt
!wget -q https://www.dropbox.com/s/3gki5n5djs9w0v6/gazeta_test.txt
```

```
!pip install -Uq razdel allennlp torch fasttext OpenNMT-py networkx pymorphy2 nl
!pip install -Uq transformers youtokentome
```

```
     |████████████████████████████████| 512kB 11.1MB/s
     |████████████████████████████████| 71kB 7.3MB/s
     |████████████████████████████████| 204kB 16.1MB/s
     |████████████████████████████████| 61kB 7.8MB/s
     |████████████████████████████████| 1.4MB 18.8MB/s
     |████████████████████████████████| 61kB 9.9MB/s
     |████████████████████████████████| 266kB 43.8MB/s
     |████████████████████████████████| 1.3MB 53.3MB/s
     |████████████████████████████████| 133kB 54.6MB/s
     |████████████████████████████████| 317kB 57.8MB/s
     |████████████████████████████████| 61kB 9.0MB/s
     |████████████████████████████████| 51kB 6.3MB/s
     |████████████████████████████████| 61kB 9.3MB/s
     |████████████████████████████████| 2.5MB 52.5MB/s
     |████████████████████████████████| 8.2MB 46.0MB/s
     |████████████████████████████████| 890kB 54.6MB/s
     |████████████████████████████████| 2.9MB 42.0MB/s
     |████████████████████████████████| 1.1MB 54.7MB/s
     |████████████████████████████████| 7.0MB 53.9MB/s
     |████████████████████████████████| 71kB 11.0MB/s
  Building wheel for fasttext (setup.py) ... done
```

```
  Building wheel for nltk (setup.py) ... done
  Building wheel for summa (setup.py) ... done
  Building wheel for overrides (setup.py) ... done
  Building wheel for jsonnet (setup.py) ... done
  Building wheel for configargparse (setup.py) ... done
  Building wheel for sacremoses (setup.py) ... done
ERROR: botocore 1.19.30 has requirement urllib3<1.27,>=1.25.4; python_version
     |████████████████████████████████| 1.4MB 9.1MB/s
     |████████████████████████████████| 1.7MB 25.6MB/s
     |████████████████████████████████| 2.9MB 47.1MB/s
ERROR: allennlp 1.2.2 has requirement transformers<3.6,>=3.4, but you'll have
```

```
import random
import pandas as pd

def read_gazeta_records(file_name, shuffle=True, sort_by_date=False):
    assert shuffle != sort_by_date
    records = []
    with open(file_name, "r") as r:
        for line in r:
            records.append(eval(line)) # Simple hack
    records = pd.DataFrame(records)
    if sort_by_date:
        records = records.sort("date")
    if shuffle:
        records = records.sample(frac=1)
    return records
```

```
train_records = read_gazeta_records("gazeta_train.txt")
val_records = read_gazeta_records("gazeta_val.txt")
test_records = read_gazeta_records("gazeta_test.txt")
```

```
train_records.head()
```

	url	text	title
23366	https://www.gazeta.ru/social/2017/10/11/109281...	О том, что чиновники Роскомнадзор а стали фигу...	Следователи пришли в Роскомнадзор
14768	https://www.gazeta.ru/comments/2011/06/07_e_36...	Кремлевская администрация и правительство пыта...	Сумеречное решение

```
train_records.shape, val_records.shape, test_records.shape
```

```
((52400, 5), (5265, 5), (5770, 5))
```

1 задание: TextRank (порог: 0.35 BLEU)

TextRank - unsupervised метод для составления кратких выжимок из текста. Описание метода:

1. Сплитим текст по предложениям
2. Считаем "похожесть" предложений между собой
3. Строим граф предложений с взвешенными ребрами
4. С помощью алгоритм PageRank получаем наиболее важные предложения, на основе которых делаем summary.

Функция похожести можно сделать и из нейросетевых(или около) моделек: FastText, ELMO и BERT. Выберете один метод, загрузите предобученную модель и с ее помощью для каждого предложениия сделайте sentence embedding. С помощью косинусной меры определяйте похожесть предложений.

Предобученные модели можно взять по ссылке.

```
from nltk.translate.bleu_score import corpus_bleu
from rouge import Rouge

def calc_scores(references, predictions, metric="all"):
    print("Count:", len(predictions))
    print("Ref:", references[-1])
    print("Hyp:", predictions[-1])

    if metric in ("bleu", "all"):
        print("BLEU: ", corpus_bleu([[r] for r in references], predictions))
    if metric in ("rouge", "all"):
        rouge = Rouge()
        scores = rouge.get_scores(predictions, references, avg=True)
        print("ROUGE: ", scores)
```

```
!pip install catalyst
```

```
Requirement already satisfied: pandas>=0.22 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: ipython in /usr/local/lib/python3.6/dist-packa
Collecting deprecation
  Downloading https://files.pythonhosted.org/packages/02/c3/253a89ee03fc9b968
Requirement already satisfied: PyYAML in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: numpy>=1.16.4 in /usr/local/lib/python3.6/dist
Requirement already satisfied: tensorboardX in /usr/local/lib/python3.6/dist-
Requirement already satisfied: tqdm>=4.33.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: packaging in /usr/local/lib/python3.6/dist-pac

Requirement already satisfied: torch>=1.1.0 in /usr/local/lib/python3.6/dist-
Collecting gitdb<5,>=4.0.1
  Downloading https://files.pythonhosted.org/packages/48/11/d1800bca0a3bae820
     |████████████████████████████████| 71kB 8.1MB/s
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /u
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
```

```
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/di
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/di
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/li
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.6
Requirement already satisfied: google-auth<2,>=1.6.3 in /usr/local/lib/python
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.
Requirement already satisfied: grpcio>=1.24.3 in /usr/local/lib/python3.6/dis
Requirement already satisfied: wheel>=0.26; python_version >= "3" in /usr/loc
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: protobuf>=3.6.0 in /usr/local/lib/python3.6/di
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.6/
Requirement already satisfied: pickleshare in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: decorator in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: pexpect; sys_platform != "win32" in /usr/local
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.6/dis
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib
Requirement already satisfied: pygments in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.6/
Requirement already satisfied: dataclasses in /usr/local/lib/python3.6/dist-p
Collecting smmap<4,>=3.0.1
  Downloading https://files.pythonhosted.org/packages/b0/9a/4d409a6234eb940e6
Requirement already satisfied: importlib-metadata; python_version < "3.8" in
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/pytho
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python
Requirement already satisfied: rsa<5,>=3.1.4; python_version >= "3" in /usr/l
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/pyt
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.6/di
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.6/d
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.6/di
```

```
1 import torch
2 import transformers
3 from tqdm import tqdm
4 from sklearn.metrics.pairwise import cosine_similarity
5 from catalyst.utils import set_global_seed
6
7 seed = 42
8 set_global_seed(seed)
```

```
1 cosine_similarity([[1,2,3,4,5]], [[2,3,4,5,6]]), cosine_similarity([[1,2,3,4,5]]
```

```
(array([[0.99493668]]), array([[0.63636364]]), array([[-1.]]))
```

```
1 # загрузим предварительно обученные модели лля токенизации и получения эмбеддингов
2
3 # pretrained_model_name = "DeepPavlov/rubert-base-cased-sentence"
4 # tokenizer = transformers.AutoTokenizer.from_pretrained(pretrained_model_name)
```

```
# model = transformers.AutoModelWithLMHead.from_pretrained(pretrained_model_name
```

```
# !wget http://files.deeppavlov.ai/deeppavlov_data/bert/sentence_ru_cased_L-12_H
```

```
--2020-12-06 17:08:46--  http://files.deeppavlov.ai/deeppavlov_data/bert/sente
Resolving files.deeppavlov.ai (files.deeppavlov.ai)... 93.175.29.74
Connecting to files.deeppavlov.ai (files.deeppavlov.ai)|93.175.29.74|:80... co
HTTP request sent, awaiting response... 200 OK
Length: 661614603 (631M) [application/octet-stream]
Saving to: 'sentence_ru_cased_L-12_H-768_A-12_pt.tar.gz'

sentence_ru_cased_L 100%[===================>] 630.96M  3.02MB/s    in 3m 11s

2020-12-06 17:11:59 (3.31 MB/s) - 'sentence_ru_cased_L-12_H-768_A-12_pt.tar.g
```

```
# !tar -xvzf sentence_ru_cased_L-12_H-768_A-12_pt.tar.gz
```

```
sentence_ru_cased_L-12_H-768_A-12_pt/
sentence_ru_cased_L-12_H-768_A-12_pt/pytorch_model.bin
sentence_ru_cased_L-12_H-768_A-12_pt/bert_config.json
sentence_ru_cased_L-12_H-768_A-12_pt/vocab.txt
```

```
!wget http://files.deeppavlov.ai/deeppavlov_data/bert/sentence_multi_cased_L-12_
```

```
--2020-12-06 17:44:28--  http://files.deeppavlov.ai/deeppavlov_data/bert/sente
Resolving files.deeppavlov.ai (files.deeppavlov.ai)... 93.175.29.74
Connecting to files.deeppavlov.ai (files.deeppavlov.ai)|93.175.29.74|:80... co
HTTP request sent, awaiting response... 200 OK
Length: 662398225 (632M) [application/octet-stream]
Saving to: 'sentence_multi_cased_L-12_H-768_A-12_pt.tar.gz'

sentence_multi_case 100%[===================>] 631.71M  5.53MB/s    in 3m 17s

2020-12-06 17:47:46 (3.21 MB/s) - 'sentence_multi_cased_L-12_H-768_A-12_pt.ta
```

```
!tar -xvzf sentence_multi_cased_L-12_H-768_A-12_pt.tar.gz
```

```
sentence_multi_cased_L-12_H-768_A-12_pt/
sentence_multi_cased_L-12_H-768_A-12_pt/pytorch_model.bin
sentence_multi_cased_L-12_H-768_A-12_pt/bert_config.json
sentence_multi_cased_L-12_H-768_A-12_pt/vocab.txt
```

```
import transformers
import torch
class Seq2Vec():
  def __init__(self,vocabName="vocab.txt",bertConfig = "bert_config.json",bertNa
    #   https://huggingface.co/transformers/pretrained_models.html
    self.tokenizer = transformers.BertTokenizer.from_pretrained(tokenizer_name)
    self.simple_vocab = {}
    with open(vocabName,"r") as vocab:
      all_text = vocab.read().split('\n')
      for i in range(len(all_text)):
        self.simple_vocab[all_text[i]] = i
```

```
    self.model = transformers.BertModel.from_pretrained(bertName, config = bert(
    self.model.to("cuda")
    self.model.eval()
    self.data = {}
    self.PAD = self.simple_vocab['[PAD]']
    self.SEP = self.simple_vocab['[SEP]']
    self.CLS = self.simple_vocab['[CLS]']
  def get_ids(self,tokens):
    result = [self.CLS]
    for k in tokens:
      if (k in self.simple_vocab):
         result.append(self.simple_vocab[k])
      else: result.append(0)
    result.append(self.SEP)
    if (result == [self.CLS]): return [self.CLS,0,self.SEP]
    return result
  def get_vec(self,sequence):
    with torch.no_grad():
      if("|".join(sequence) in self.data):
        return self.data["|".join(sequence)]
      tokens = self.tokenizer.tokenize(" ".join(sequence))
      tokens = self.get_ids(tokens)

      embeddings = self.model(torch.LongTensor([tokens]).to("cuda")).last_hidden

      res = embeddings.cpu().reshape(-1).detach().numpy()
      self.data["|".join(sequence)] = res
      return res

model = Seq2Vec()
```

```
def embed_str(sentence):
    '''
    Функция для вывода массива эмбедингов предложения, полученных с пом.
    модели BERT. Эти эмбединги - выход скрытого слоя модели.
    '''
    # print(sentence)
    if len(sentence) > 1:
        sentence = ' '.join(sentence)
    # if sentence in embed_sentence:
    #     return embed_sentence[sentence]

    input_ids = torch.tensor(tokenizer.encode(sentence)).unsqueeze(0)
    outputs = model(input_ids)
    last_hidden_states = [outputs[0].detach().numpy().squeeze()[-1, :]]
    # embed_sentence[sentence] = last_hidden_states
    return last_hidden_states
```

```
from itertools import combinations
import networkx as nx
import numpy as np
import pymorphy2
import razdel

```

```
i = 0

def unique_words_similarity(words1, words2):
    '''
    Функция подсчёта близости предложений на основе пересечения слов
    '''
    global i
    i += 1
    words1 = set(words1)
    words2 = set(words2)
    if not len(words1) or not len(words2):
        return 0.0
    return len(words1.intersection(words2))/(np.log10(len(words1)) + np.log10(le


def your_super_words_similarity(words1, words2):
    global i
    i += 1
    # print(i)

    # words1 = np.squeeze(words1)
    # words2 = np.squeeze(words2)
    # print('your_super_words_similarity')
    words1, words2 = model.get_vec(words1), model.get_vec(words2)
    # print(words1)
    # print(words1.shape)
    # print(words2)
    # print(words2.shape)
    dist = cosine_similarity([words1], [words2]) # [:, 0]
    # print('dist', dist)
    return dist[0]
    # words1 = words1/np.linalg.norm(words1)
    # words2 = words2/np.linalg.norm(words2)

    # return words1.dot(words2)


def gen_text_rank_summary(text, calc_similarity=unique_words_similarity, summary
    '''
    Составление summary с помощью TextRank
    '''
    # Разбиваем текст на предложения
    sentences = [sentence.text for sentence in razdel.sentenize(text)]
    n_sentences = len(sentences)

    # Токенизируем предложения
    sentences_words = [[token.text.lower() if lower else token.text for token ir

    # При необходимости лемматизируем слова
    if morph is not None:
        sentences_words = [[morph.parse(word)[0].normal_form for word in words]

    # Для каждой пары предложений считаем близость
    pairs = combinations(range(n_sentences), 2)
    # print('len pairs ', len(pairs))
```

```
    # if calc_similarity==unique_words_similarity:
    scores = [(i, j, calc_similarity(sentences_words[i], sentences_words[j])) fo
    # else:
    #     '''
    #     скор для варианта с рассчетом косинусного расстояния м-ду эмбеддингами
    #     переведём sentences_words в эмбединги
    #     '''
    #     scores = [(i, j, calc_similarity(embed_str(sentences_words[i]), embed_

    print('i = ', i)
    # Строим граф с рёбрами, равными близости между предложениями
    g = nx.Graph()
    g.add_weighted_edges_from(scores)

    # Считаем PageRank
    pr = nx.pagerank(g)
    result = [(i, pr[i], s) for i, s in enumerate(sentences) if i in pr]
    result.sort(key=lambda x: x[1], reverse=True)

    # Выбираем топ предложений
    n_summary_sentences = max(int(n_sentences * summary_part), 1)
    result = result[:n_summary_sentences]

    # Восстанавливаем оригинальный их порядок
    result.sort(key=lambda x: x[0])

    # Восстанавливаем текст выжимки
    predicted_summary = " ".join([sentence for i, proba, sentence in result])
    predicted_summary = predicted_summary.lower() if lower else predicted_summar
    return predicted_summary

def calc_text_rank_score(records, calc_similarity=unique_words_similarity, summa
    references = []
    predictions = []

    for text, summary in records[['text', 'summary']].values[:nrows]:
        summary = summary if not lower else summary.lower()
        references.append(summary)

        predicted_summary = gen_text_rank_summary(text, calc_similarity, summary
        text = text if not lower else text.lower()
        predictions.append(predicted_summary)

    calc_scores(references, predictions)
```

```
# test_records
```

```
# calc_text_rank_score(test_records, calc_similarity=unique_words_similarity)
```

```
calc_text_rank_score(test_records, calc_similarity=your_super_words_similarity)
```

```
i =  583934
i =  584124
i =  584790
i =  586065
```

```
i =  586065
i =  587193
i =  587518
i =  587924
i =  588452
i =  589313
i =  589908
i =  590404
i =  591034
i =  591737
i =  592202
i =  592553
i =  593049
i =  593325
i =  593920
i =  594866
i =  595427
i =  596168
i =  596696
i =  597399
i =  597750
i =  598026
i =  598491
i =  599157
i =  599653
i =  601423
i =  601633
i =  602374
i =  603004

i =  603599
i =  604502
i =  604967
i =  605292
i =  605958
i =  607086
i =  607362
i =  607858
i =  608599
i =  609005
i =  609440
i =  609650
i =  610028
i =  610889
i =  611592
i =  612088
i =  613414
i =  614080
i =  615971
i =  616322
i =  616757
i =  617352
i =  617505
i =  618066
i =  618732
i =  619197
i =  620523
i =  621189
```

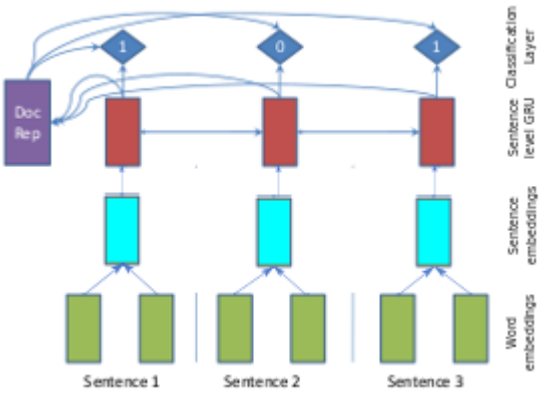
2 Задание: Extractive RNN (порог: 0.35 BLEU)

Второй метод, который вам предлагается улучшить – поиск предложений для summary с помощью RNN. В рассмотренной методе мы использовали LSTM для генерации sentence embedding. Попробуйте использовать другие архитектуры: CNN, Transformer; или добавьте предобученные модели, как и в первом задании.

P.S. Тут предполагается, что придется изменять много кода в ячеeках (например, поменять токенизацию).

Модель

Картинка для привлечения внимания:



Статья с оригинальным методом: https://arxiv.org/pdf/1611.04230.pdf

Список вдохновения:

- https://towardsdatascience.com/understanding-how-convolutional-neural-network-cnn-perform-text-classification-with-word-d2ee64b9dd0b Пример того, как можно применять CNN в текстовых задачах
- https://arxiv.org/pdf/1808.08745.pdf Очень крутой метод генерации summary без Transformers
- https://towardsdatascience.com/super-easy-way-to-get-sentence-embedding-using-fasttext-in-python-a70f34ac5b7c – простой метод генерации sentence embedding
- https://towardsdatascience.com/fse-2b1ffa791cf9 – Необычный метод генерации sentence embedding
- https://github.com/UKPLab/sentence-transformers – BERT предобученный для sentence embedding

P.S. Выше написанные ссылки нужны только для разогрева вашей фантазии, можно воспользоваться ими, а можно придумать свой.

Комментарий к заданию: Если посмотреть на архитектуру ~~почти~~ SummaRuNNer, то в ней есть два главных элемента: первая часть, которая читает предложения и возвращает векторы на каждое предложение, и вторая, которая выбирает предложения для суммаризации. Вторую часть мы не трогаем, а первую меняем. На что меняем – как вы решите. Главное: она должна иметь хорошее качество и встроиться в текущую модель.

```
import copy
import random

def build_oracle_summary_greedy(text, gold_summary, calc_score, lower=True, max_
    '''
    Жадное построение oracle summary
    '''
    gold_summary = gold_summary.lower() if lower else gold_summary
    # Делим текст на предложения
    sentences = [sentence.text.lower() if lower else sentence.text for sentence
    n_sentences = len(sentences)
    oracle_summary_sentences = set()
    score = -1.0
    summaries = []
    for _ in range(min(n_sentences, 2)):
        for i in range(n_sentences):
            if i in oracle_summary_sentences:
                continue
            current_summary_sentences = copy.copy(oracle_summary_sentences)
            # Добавляем какое-то предложения к уже существующему summary
            current_summary_sentences.add(i)
            current_summary = " ".join([sentences[index] for index in sorted(lis
            # Считаем метрики
            current_score = calc_score(current_summary, gold_summary)
            summaries.append((current_score, current_summary_sentences))
        # Если получилось улучшить метрики с добавлением какого-либо предложения, то пробуем
        # Иначе на этом заканчиваем
        best_summary_score, best_summary_sentences = max(summaries)
        if best_summary_score <= score:
            break
        oracle_summary_sentences = best_summary_sentences
        score = best_summary_score
    oracle_summary = " ".join([sentences[index] for index in sorted(list(oracle_
    return oracle_summary, oracle_summary_sentences

def calc_single_score(pred_summary, gold_summary, rouge):
    return rouge.get_scores([pred_summary], [gold_summary], avg=True)['rouge-2']
```

```
from tqdm import tqdm_notebook as tqdm

def calc_oracle_score(records, nrows=1000, lower=True):
    references = []
    predictions = []
    rouge = Rouge()

    for text, summary in tqdm(records[['text', 'summary']].values[:nrows]):
        summary = summary if not lower else summary.lower()
        references.append(summary)
        predicted_summary, _ = build_oracle_summary_greedy(text, summary, calc_s
        predictions.append(predicted_summary)

    calc_scores(references, predictions)

```

```
16 calc_oracle_score(test_records)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: TqdmDeprecati

This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`

100%                                                  1000/1000 [02:26<00:00, 6.80it/s]

Count: 1000
Ref: челябинскую городскую больницу №6 из-за мизерных зарплат продолжают покидать врачи. 
Hyp: однако руководство больницы никак не реагирует на происходящее. тем временем в челябин
BLEU:  0.4340029354374597
ROUGE:  {'rouge-1': {'f': 0.3539652034259989, 'p': 0.42831485382882784, 'r':
```

(!)

Если надо, поменяйте код загрузки токенизатора

```
 1 import os
 2
 3 import youtokentome as yttm
 4
 5 def train_bpe(records, model_path, model_type="bpe", vocab_size=30000, lower=Tru
 6     temp_file_name = "temp.txt"
 7     with open(temp_file_name, "w") as temp:
 8         for text, summary in records[['text', 'summary']].values:
 9             if lower:
10                 summary = summary.lower()
11                 text = text.lower()
12             if not text or not summary:
13                 continue
14             temp.write(text + "\n")
15             temp.write(summary + "\n")
16     yttm.BPE.train(data=temp_file_name, vocab_size=vocab_size, model=model_path)
17
18 train_bpe(train_records, "BPE_model.bin")
```

```
 1 bpe_processor = yttm.BPE('BPE_model.bin')
 2 bpe_processor.encode(["октябрь богат на изменения"], output_type=yttm.OutputType.SUB
```

```
[['▁октябрь', '▁богат', '▁на', '▁изменения']]
```

(!)

Если надо, поменяйте код словаря

```
1 from collections import Counter
2 from typing import List, Tuple
```

```
import os

class Vocabulary:
    def __init__(self, bpe_processor):
        self.index2word = bpe_processor.vocab()
        self.word2index = {w: i for i, w in enumerate(self.index2word)}
        self.word2count = Counter()

    def get_pad(self):
        return self.word2index["<PAD>"]

    def get_sos(self):
        return self.word2index["<SOS>"]

    def get_eos(self):
        return self.word2index["<EOS>"]

    def get_unk(self):
        return self.word2index["<UNK>"]

    def has_word(self, word) -> bool:
        return word in self.word2index

    def get_index(self, word):
        if word in self.word2index:
            return self.word2index[word]
        return self.get_unk()

    def get_word(self, index):
        return self.index2word[index]

    def size(self):
        return len(self.index2word)

    def is_empty(self):
        empty_size = 4
        return self.size() <= empty_size

    def reset(self):
        self.word2count = Counter()
        self.index2word = ["<pad>", "<sos>", "<eos>", "<unk>"]
        self.word2index = {word: index for index, word in enumerate(self.index2v
```

```
vocabulary = Vocabulary(bpe_processor)
vocabulary.size()
```

```
30000
```

```
from rouge import Rouge
import razdel

def add_oracle_summary_to_records(records, max_sentences=30, lower=True, nrows=1
    rouge = Rouge()
    sentences_ = []
```

```
    oracle_sentences_ = []
    oracle_summary_ = []
    if nrows is not None:
        records = records.iloc[:nrows].copy()
    else:
        records = records.copy()

    for text, summary in tqdm(records[['text', 'summary']].values):
        summary = summary.lower() if lower else summary
        sentences = [sentence.text.lower() if lower else sentence.text for sente
        oracle_summary, sentences_indicies = build_oracle_summary_greedy(text, s
                                                                         lower=l
        sentences_ += [sentences]
        oracle_sentences_ += [list(sentences_indicies)]
        oracle_summary_ += [oracle_summary]
    records['sentences'] = sentences_
    records['oracle_sentences'] = oracle_sentences_
    records['oracle_summary'] = oracle_summary_
    return records

ext_train_records = add_oracle_summary_to_records(train_records, nrows=30000)
ext_val_records = add_oracle_summary_to_records(val_records, nrows=None)
ext_test_records = add_oracle_summary_to_records(test_records, nrows=None)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:14: TqdmDeprecat

This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`

100%                                      30000/30000 [29:04<00:00, 17.19it/s]


100%                                      5265/5265 [06:01<00:00, 14.58it/s]


100%                                      5770/5770 [05:59<00:00, 16.03it/s]
```

Используй `pickle` для сохранения записей, чтобы потом не пересоздавать их потом. Если решаешь задание в колабе, можешь подключить свой гугл диск и сохранить данные в нём.

```

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

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

```
import pickle

with open("train_records.bin", 'wb') as file:
    pickle.dump(ext_train_records, file)

```

```
with open("val_records.bin", 'wb') as file:
    pickle.dump(ext_val_records, file)

with open("test_records.bin", 'wb') as file:
    pickle.dump(ext_test_records, file)
```

```
# pickle_train = open("/content/drive/MyDrive/train_records.bin","rb")
# pickle_test = open("/content/drive/MyDrive/test_records.bin","rb")
# pickle_valid = open("/content/drive/MyDrive/val_records.bin","rb")

# ext_test_records  = pickle.load(pickle_test)
# ext_val_records = pickle.load(pickle_valid)
# ext_train_records = pickle.load(pickle_train)
```

(!)

Если надо, поменяйте код генератора датасета и батчевалки

```
import random
import math
import razdel
import torch
import numpy as np
from rouge import Rouge


from torch.utils import data


class ExtDataset(data.Dataset):
    def __init__(self, records, vocabulary, bpe_processor, lower=True, max_sente
        self.records = records
        self.num_samples = records.shape[0]
        self.bpe_processor = bpe_processor
        self.lower = lower
        self.rouge = Rouge()
        self.vocabulary = vocabulary
        self.max_sentences = max_sentences
        self.max_sentence_length = max_sentence_length
        self.device = device

    def __len__(self):
        return self.records.shape[0]

    def __getitem__(self, idx):
        cur_record = self.records.iloc[idx]
        inputs = list(map(lambda x: x[:self.max_sentence_length], self.bpe_proce
        outputs = [int(i in cur_record['oracle_sentences']) for i in range(len(c
        return {'inputs': inputs, 'outputs': outputs}
```

```
# Это батчевалка
def collate_fn(records):
```

```
def collate_fn(records):
    max_length = max(len(sentence) for record in records for sentence in record[
    max_sentences = max(len(record['outputs']) for record in records)

    new_inputs = torch.zeros((len(records), max_sentences, max_length))
    new_outputs = torch.zeros((len(records), max_sentences))
    for i, record in enumerate(records):
        for j, sentence in enumerate(record['inputs']):
            new_inputs[i, j, :len(sentence)] += np.array(sentence)
        new_outputs[i, :len(record['outputs'])] += np.array(record['outputs'])
    return {'features': new_inputs.type(torch.LongTensor), 'targets': new_output
```

```
import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from torch.nn.utils.rnn import pack_padded_sequence as pack
from torch.nn.utils.rnn import pad_packed_sequence as unpack


class YourSentenceEncoder(nn.Module):
    # Место для вашего Sentence Encoder-а. Разрешается использовать любые методы, которые
    def __init__(self, input_size, embedding_dim, hidden_size, n_layers=3, dropo
        super().__init__()

        num_directions = 2 if bidirectional else 1
        assert hidden_size % num_directions == 0
        hidden_size = hidden_size // num_directions

        self.embedding_dim = embedding_dim
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.n_layers = n_layers
        self.dropout = dropout
        self.bidirectional = bidirectional

        self.embedding_layer = nn.Embedding(input_size, embedding_dim)
        self.rnn_layer = nn.LSTM(embedding_dim, hidden_size, n_layers, dropout=d
        self.dropout_layer = nn.Dropout(dropout)

    def forward(self, inputs, hidden=None):
        embedded = self.embedding_layer(inputs)
        outputs, _ = self.rnn_layer(embedded, hidden)
        sentences_embeddings = torch.mean(outputs, 1)

        return sentences_embeddings


class SentenceTaggerRNN(nn.Module):
    def __init__(self,
                 vocabulary_size,
                 token_embedding_dim=256,
```

```
                 sentence_encoder_hidden_size=256,
                 hidden_size=256,
                 bidirectional=True,
                 sentence_encoder_n_layers=2,
                 sentence_encoder_dropout=0.3,
                 sentence_encoder_bidirectional=True,
                 n_layers=1,
                 dropout=0.3):
        super(SentenceTaggerRNN, self).__init__()

        num_directions = 2 if bidirectional else 1
        assert hidden_size % num_directions == 0
        hidden_size = hidden_size // num_directions

        self.hidden_size = hidden_size
        self.n_layers = n_layers
        self.dropout = dropout
        self.bidirectional = bidirectional

        # Your sentence encoder model
        self.sentence_encoder = YourSentenceEncoder(vocabulary_size, token_embed
                                                    sentence_encoder_hidden_size,
                                                    sentence_encoder_dropout, ser

        self.rnn_layer = nn.LSTM(
            sentence_encoder_hidden_size,
            hidden_size,
            n_layers,
            dropout=dropout,
            bidirectional=bidirectional,
            batch_first=True)
        self.dropout_layer = nn.Dropout(dropout)
        self.content_linear_layer = nn.Linear(hidden_size * 2, 1)
        self.document_linear_layer = nn.Linear(hidden_size * 2, hidden_size * 2)
        self.salience_linear_layer = nn.Linear(hidden_size * 2, hidden_size * 2)
        self.tanh_layer = nn.Tanh()

    def forward(self, inputs, hidden=None):
        batch_size = inputs.size(0)
        sentences_count = inputs.size(1)
        tokens_count = inputs.size(2)
        inputs = inputs.reshape(-1, tokens_count)
        embedded_sentences = self.sentence_encoder(inputs)
        embedded_sentences = embedded_sentences.reshape(batch_size, sentences_co
        outputs, _ = self.rnn_layer(embedded_sentences, hidden)
        outputs = self.dropout_layer(outputs)
        document_embedding = self.tanh_layer(self.document_linear_layer(torch.me
        content = self.content_linear_layer(outputs).squeeze(2)
        salience = torch.bmm(outputs, self.salience_linear_layer(document_embedd
        return content + salience

model = SentenceTaggerRNN(vocabulary.size())
```

```
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/rnn.py:61: UserWarnin
```

```
    dropout option adds dropout after all but last recurrent layer, so non-zero d
```

Обучение

```
device = torch.device('cuda')

loaders = {
    'train': data.DataLoader(
        ExtDataset(
            ext_train_records,
            vocabulary,
            bpe_processor=bpe_processor
        ),
        batch_size=64,
        collate_fn=collate_fn
    ),
    'valid': data.DataLoader(
        ExtDataset(
            ext_val_records,
            vocabulary,
            bpe_processor=bpe_processor
        ),
        batch_size=64,
        collate_fn=collate_fn
    ),
    'test': data.DataLoader(
        ExtDataset(
            ext_test_records,
            vocabulary,
            bpe_processor=bpe_processor
        ),
        batch_size=64,
        collate_fn=collate_fn
    ),
}

lr = 1e-4
num_epochs = 5

optimizer  = torch.optim.Adam(model.parameters(), lr=lr)
criterion = nn.BCEWithLogitsLoss()
# Maybe adding scheduler?
```

```
from tqdm.notebook import trange, tqdm


def train():
    model.to(device)
    pbar_loader = trange(len(loaders["train"]) + len(loaders["valid"]), desc=f"T
    for e in trange(num_epochs, desc="Epoch"):
        train_loss = 0
        valid_loss = 0
```

```
    valid_loss = 0
    train_it = 0
    valid_it = 0

    model.train()
    for batch in loaders["train"]:
        features = batch["features"].to(device)
        targets = batch["targets"].to(device)

        logits = model(features)

        loss = criterion(logits, targets)
        train_loss += loss.item()
        train_it += 1

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # Maybe adding scheduler?

        pbar_loader.update()
        pbar_loader.set_description(
            f"Train Loss: {train_loss / train_it:.3}, Valid Loss: {0}"
        )

    model.eval()
    with torch.no_grad():
        for batch in loaders["valid"]:
            features = batch["features"].to(device)
            targets = batch["targets"].to(device)

            logits = model(features)

            loss = criterion(logits, targets)
            valid_loss += loss.item()
            valid_it += 1

            pbar_loader.update()
            pbar_loader.set_description(
                f"Train Loss: {train_loss / train_it:.3},"
                f" Valid Loss: {valid_loss / valid_it:.3}"
            )
    print(
        f"Epoch {e}; Train Loss: {train_loss / train_it:.3},"
        f" Valid Loss: {valid_loss / valid_it:.3}"
    )
    pbar_loader.reset()
```

```
train()
```

Train Loss: 0.178, Valid Loss: 0.179: 0% 0/552 [00:00<05:04, 1.81it/s]

Epoch: 100% 5/5 [36:18<00:00, 435.65s/it]

```
Epoch 0; Train Loss: 0.204, Valid Loss: 0.183
Epoch 1; Train Loss: 0.189, Valid Loss: 0.181
```

```
device = torch.device("cuda")

top_k = 3

def postprocess(ref, hyp, is_multiple_ref=False, detokenize_after=False, tokeniz
    if is_multiple_ref:
        reference_sents = ref.split(" s_s ")
        decoded_sents = hyp.split("s_s")
        hyp = [w.replace("<", "&lt;").replace(">", "&gt;").strip() for w in deco
        ref = [w.replace("<", "&lt;").replace(">", "&gt;").strip() for w in refe
        hyp = " ".join(hyp)
        ref = " ".join(ref)
    ref = ref.strip()
    hyp = hyp.strip()
    if detokenize_after:
        hyp = punct_detokenize(hyp)
        ref = punct_detokenize(ref)
    if tokenize_after:
        hyp = hyp.replace("@@UNKNOWN@@", "<unk>")
        hyp = " ".join([token.text for token in razdel.tokenize(hyp)])
        ref = " ".join([token.text for token in razdel.tokenize(ref)])
    return ref, hyp

references = []
predictions = []

model.eval()
for num, batch in enumerate(loaders["test"]):

    logits = model(batch["features"].to(device))
    in_summary = torch.argsort(logits, dim=1)[:, -top_k:]
    for i in range(len(batch['targets'])):

        summary = ext_test_records.iloc[i]['summary']
        summary = summary.lower()
        predicted_summary = ' '.join([ext_test_records.iloc[i]['sentences'][idx]
        summary, predicted_summary = postprocess(summary, predicted_summary)

        references.append(summary)
        predictions.append(predicted_summary)

calc_scores(references, predictions)
```

```
Count: 5770
Ref: следователи возбудили дело против жительницы сургута , подозреваемой в убийстве 49-лет
Hyp: жительницу сургута подозревают в убийстве 49-летнего сожителя . по версии ведомства ,
BLEU:  0.4482006046197879
ROUGE:  {'rouge-1': {'f': 0.30057638825676386, 'p': 0.28523669319955836, 'r':
```

⊥