Введение в обработку естественного языка

Урок 3. Embedding word2vec fasttext

Задача поиск похожих по эмбеддингам

Скачиваем датасет (источник - http://study.mokoron.com/ (http://study.mokoron.com/)):

- положительные - https://www.dropbox.com/s/fnpq3z4bcnoktiv/positive.csv?dl=0 (https://www.dropbox.com/s/fnpq3z4bcnoktiv/positive.csv?dl=0),
- отрицательные - https://www.dropbox.com/s/r6u59ljhhjdg6j0/negative.csv (https://www.dropbox.com/s/r6u59ljhhjdg6j0/negative.csv).

или можно через ноутбук

!wget https://www.dropbox.com/s/fnpq3z4bcnoktiv/positive.csv (https://www.dropbox.com/s/fnpq3z4bcnoktiv/positive.csv)
!wget https://www.dropbox.com/s/r6u59ljhhjdg6j0/negative.csv (https://www.dropbox.com/s/r6u59ljhhjdg6j0/negative.csv)

что надо сделать

1. объединить в одну выборку
2. на основе word2vec/fasttext/glove/слоя Embedding реализовать метод поиска ближайших твитов
   (на вход метода должен приходить запрос (какой-то твит, вопрос) и количество вариантов вывода к примеру 5-ть, ваш метод должен возвращать 5-ть ближайших твитов к этому запросу)
3. Проверить насколько хорошо работают подходы

Выполнил ***Соковнин ИЛ***

В [1]:
```
import pandas as pd
import numpy as np
import re
from sklearn.metrics import *
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
```

В [2]:
```
# Сброс ограничений на количество символов в записи
pd.set_option('display.max_colwidth', None)
```

Задания:

1. Скачать датасет и объединить в одну выборку

В [3]:
```
# Как объединить DataFrames в Pandas - merge (), join (), append (), concat () и update ()
# https://rukovodstvo.net/posts/id_606/#mergedataframesusingappend
# считываем данные и заполняем общий датасет
positive = pd.read_csv('./data/positive.csv', sep=';', usecols=[3], names=['text'])
negative = pd.read_csv('./data/negative.csv', sep=';', usecols=[3], names=['text'])
df = positive.append(negative)
df
```

Out[3]:

	text
0	@first_timee хоть я и школота, но поверь, у нас то же самое :D общество профилирующий предмет типа)
1	Да, все-таки он немного похож на него. Но мой мальчик все равно лучше:D
2	RT @KatiaCheh: Ну ты идиотка) я испугалась за тебя!!!
3	RT @digger2912: "Кто то в углу сидит и погибает от голода, а мы ещё 2 порции взяли, хотя уже и так жрать не хотим" :DD http://t.co/GqG6iuE2…
4	@irina_dyshkant Вот что значит страшилка :D\nНо блин,посмотрев все части,у тебя создастся ощущение,что авторы курили что-то :D
...	...
111918	Но не каждый хочет что то исправлять:(http://t.co/QNODDQzuZ7
111919	скучаю так :-(только @taaannyaaa вправляет мозги, но я все равно скучаю
111920	Вот и в школу, в говно это идти уже надо(
111921	RT @_Them__: @LisaBeroud Тауриэль, не грусти :(*обнял*
111922	Такси везет меня на работу. Раздумываю приплатить, чтобы меня втащили на пятый этаж. Лифта то нет :(

226834 rows × 1 columns

In [4]:
```
print('df', df.shape)
print('positive', positive.shape)
print('negative', negative.shape)
print(df.shape, '\n')

# print('Row count is:', df.shape[0])
# print('Row count is:', len(df.index))
# print('Row count is:', len(df.axes[0]), '\n')
```

```
df (226834, 1)
positive (114911, 1)
negative (111923, 1)
(226834, 1)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 226834 entries, 0 to 111922
Data columns (total 1 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   text    226834 non-null  object
dtypes: object(1)
memory usage: 3.5+ MB
```

In [6]:
```
# positive.tail()
```

In [7]:
```
# negative.tail()
```

In [8]:
```
# Выбрать конкретную строку и столбец
df.iloc[114910, ]
```

Out[8]:
```
text    @Ma_che_rie посмотри #непытайтесьпокинутьомск сегодня в Вавилоне в 18.20. Я там тоже есть :)
Name: 114910, dtype: object
```

In [9]:
```
df.iloc[111922, ]
```

Out[9]:
```
text    RT @bazzzilio: @VRSoloviev :)) Я все еще жив, придурки. Перестаньте путать меня с Нельсоном Манделой. Спасибо. http://t.co/JmOp42vhwb (http://t.co/JmOp42vhwb)
Name: 111922, dtype: object
```

preprocessing

In [10]:
```
def replace_patern(text, patern, text_repl):
    """
    Заменим патерн на пробелы.
    """
    text = re.sub(patern, text_repl, text)

    return  text
```

In [11]:
```
%%time
# Удалим @user из всех текстов
df['text'] = df['text'].apply(replace_patern, patern =  r'@[\w]*', text_repl='')
# Заменим пунктуацию на пробелы, используя re.sub() и паттерн r'[^\w\s]'.
# df['text'] = df['text'].apply(replace_patern, patern =  r'[^\w\s]')
# Заменим спец. символы на пробелы
# df['text'] = df['text'].apply(replace_patern, patern =  r'[^a-zA-Z0-9]')
df.head(3)
```

```
Wall time: 846 ms
```

Out[11]:

	text
0	хоть я и школота, но поверь, у нас то же самое :D общество профилирующий предмет типа)
1	Да, все-таки он немного похож на него. Но мой мальчик все равно лучше:D
2	RT : Ну ты идиотка) я испугалась за тебя!!!

Векторизация

В [12]:
```
%%time

df.text = np.vectorize(replace_patern)(text=df.text, patern =  r'@[\w]*', text_repl='')
df.head(3)
```

```
Wall time: 1.82 s
```

Out[12]:

	text
0	хоть я и школота, но поверь, у нас то же самое :D общество профилирующий предмет типа)
1	Да, все-таки он немного похож на него. Но мой мальчик все равно лучше:D
2	RT : Ну ты идиотка) я испугалась за тебя!!!

В [13]:
```
df.to_csv("./text.txt", header=None, index=False, encoding='utf-8')  # Сохранение без индексации
```

В [14]:
```
# combine_df = pd.read_csv("./text.csv")
# combine_df.head(3)
```

В [1]:
```
%%time

text_in = open('text.txt','r', encoding='utf-8')
# print(type(text_in))
# print(*text_in)
```

```
Wall time: 983 µs
```

2. На основе word2vec/fasttext/glove/слоя Embedding реализовать метод поиска ближайших твитов

(на вход метода должен приходить запрос (какой-то твит, вопрос) и количество вариантов
вывода к примеру 5-ть, ваш метод должен возвращать 5-ть ближайших твитов к этому запросу)

Загрузка предобученной модели русского корпуса

Векторное представление слов https://neerc.ifmo.ru/wiki/index.php?title=Векторное_представление_слов (https://neerc.ifmo.ru/wiki/index.php?title=%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%B4%D1%81%D1%8

Модели https://rusvectores.org/ru/models/#ruwikiruscorpora_upos_cbow_300_10_2021 (https://rusvectores.org/ru/models/#ruwikiruscorpora_upos_cbow_300_10_2021)

Embeddings W2V https://colab.research.google.com/drive/11tkidAcwXRr8x3hDIDFd6DisVmHeTqHB?usp=sharing#scrollTo=zm9z6SN3X-rS (https://colab.research.google.com/drive/11tkidAcwXRr8x3hDIDFd6DisVmHeTqHB?usp=sharing#scrollTo=zm9z6SN3X-rS)

В [16]:
```
import gensim
import gensim.downloader as download_api
```

В [17]:
```
# список предобученных моделей
download_api.info()['models'].keys()
```

Out[17]:
```
dict_keys(['fasttext-wiki-news-subwords-300', 'conceptnet-numberbatch-17-06-300', 'word2vec-ruscorpora-300', 'word2vec-google-news-300', 'glove-wiki-gigaword-50', 'glove-wiki-gigaword-100', 'glove-wiki-gigaword-200', 'glove-wiki-gigaword-300', 'glove-twitter-25', 'glove-twitter-50', 'glove-twitter-100', 'glove-twitter-200', '__testing_word2vec-matrix-synopsis'])
```

В [18]:
```
russian_model = download_api.load('word2vec-ruscorpora-300')  # загрузим предтренированные вектора слов из gensim-data
```

In [19]:
```
# # Выведем первые 10 слов корпуса.
# # В модели "word2vec-ruscorpora-300" после слова указывается часть речи: NOUN (существительное), ADJ (прилагательное) и
# # Но существуют также предобученные модели без разделения слов по частям речи, смотри репозиторий
list(russian_model.index_to_key)[:10]
# # ['весь_DET', 'человек_NOUN', 'мочь_VERB', 'год_NOUN', 'сказать_VERB', 'время_NOUN', 'говорить_VERB', 'становиться_VE
```

Out[19]:
```
['весь_DET',
 'человек_NOUN',
 'мочь_VERB',
 'год_NOUN',
 'сказать_VERB',
 'время_NOUN',
 'говорить_VERB',
 'становиться_VERB',
 'знать_VERB',
 'самый_DET']
```

In [20]:
```
# Поиск наиболее близких по смыслу слов.
russian_model.most_similar('кошка_NOUN')
#  [('кот_NOUN', 0.7570087909698486), ('котенок_NOUN', 0.7261239290237427), ('собака_NOUN', 0.6963180303573608),
#  ('мяукать_VERB', 0.6411399841308594), ('крыса_NOUN', 0.6355636119842529), ('собачка_NOUN', 0.6092042922973633),
#  ('щенок_NOUN', 0.6028496026992798), ('мышь_NOUN', 0.5975362062454224), ('пес_NOUN', 0.5956044793128967),
#  ('кошечка_NOUN', 0.5920293927192688)]
```

Out[20]:
```
[('кот_NOUN', 0.7570087909698486),
 ('котенок_NOUN', 0.7261239290237427),
 ('собака_NOUN', 0.6963180303573608),
 ('мяукать_VERB', 0.6411399841308594),
 ('крыса_NOUN', 0.6355635523796082),
 ('собачка_NOUN', 0.6092043519020081),
 ('щенок_NOUN', 0.6028496623039246),
 ('мышь_NOUN', 0.5975363254547119),
 ('пес_NOUN', 0.5956044793128967),
 ('кошечка_NOUN', 0.5920294523239136)]
```

In [21]:
```
# Вычисление сходства слов
russian_model.similarity('мужчина_NOUN', 'женщина_NOUN')
# 0.85228276
```

Out[21]:
```
0.8522827
```

In [22]:
```
# Поиск лишнего слова
russian_model.doesnt_match('завтрак_NOUN хлопья_NOUN обед_NOUN ужин_NOUN'.split())
# хлопья_NOUN
```

Out[22]:
```
'хлопья_NOUN'
```

In [23]:
```
# Аналогия: Женщина + (Король - Мужчина) = Королева
russian_model.most_similar(positive=['король_NOUN','женщина_NOUN'], negative=['мужчина_NOUN'], topn=1)
# [('королева_NOUN', 0.7313904762268066)]
```

Out[23]:
```
[('королева_NOUN', 0.7313904166221619)]
```

In [24]:
```
# Аналогия: Франция = Париж + (Германия - Берлин)
russian_model.most_similar(positive=['париж_NOUN','германия_NOUN'], negative=['берлин_NOUN'], topn=1)
# [('франция_NOUN', 0.8673800230026245)]
```

Out[24]:
```
[('франция_NOUN', 0.8673799633979797)]
```

Загрузка предобученной модели glove

In [47]:
```
word_vectors = download_api.load("glove-wiki-gigaword-100")  # загрузим предтренированные вектора слов из gensim-data
# выведим слово наиболее близкое к 'woman', 'king' и далекое от 'man'
result = word_vectors.most_similar(positive=['woman', 'king'], negative=['man'])
print("{}: {:.4f}".format(*result[0]))
```

```
queen: 0.7699
```

In [48]:
```
# выведем лишнее слово
print(word_vectors.doesnt_match("breakfast cereal dinner lunch".split()))

print(word_vectors.doesnt_match("black green summer brown".split()))
```

```
cereal
summer
```

В [49]:
```
# определим схожесть между словами
similarity = word_vectors.similarity('woman', 'man')
print(similarity)

similarity = word_vectors.similarity('human', 'man')
print(similarity)

similarity = word_vectors.similarity('bee', 'man')
print(similarity)
```

```
0.8323494
0.5288512
0.21199904
```

В [50]:
```
# найдем top-3 самых близких слов
result = word_vectors.similar_by_word("man", topn=3)
print(result)

result = word_vectors.similar_by_word("cat", topn=3)
print(result)

result = word_vectors.similar_by_word("mouth", topn=3)
print(result)
```

```
[('woman', 0.832349419593811), ('boy', 0.7914870977401733), ('one', 0.7788748741149902)]
[('dog', 0.8798074126243591), ('rabbit', 0.7424427270889282), ('cats', 0.732300341129303)]
[('tongue', 0.7366125583648682), ('mouths', 0.687748908996582), ('ear', 0.6811771392822266)]
```

Обучение модели word2vec и fastText на текстовом корпусе

В [26]:
```
from gensim.models.word2vec import Word2Vec
from gensim.models.fasttext import FastText
import gensim.downloader as download_api
```

В [27]:
```
Word2Vec?
```

В [28]:
```
# Скачаем небольшой текстовый корпус (32 Мб) и откроем его как итерируемый набор предложений: iterable(list(string))
# В этом текстовом корпусе часть речи для слов не указывается
corpus = download_api.load('text8')
corpus
```

Out[28]:
```
<text8.Dataset at 0x2351c47d30>
```

В [29]:
```
# Обучим модели word2vec и fastText
word2vec_model = Word2Vec(corpus, vector_size=100, workers=4)
fastText_model = FastText(corpus, vector_size=100, workers=4)
```

В [30]:
```
word2vec_model.wv.most_similar('car')[:3]
# [('driver', 0.8033335208892822), ('motorcycle', 0.7368553876876831), ('cars', 0.7001584768295288)]
```

Out[30]:
```
[('driver', 0.7955479621887207),
 ('taxi', 0.7173714637756348),
 ('cars', 0.7153368592262268)]
```

В [31]:
```
fastText_model.wv.most_similar('car')[:3]
# [('lcar', 0.8733218908309937), ('boxcar', 0.8559106588363647), ('ccar', 0.8268736004829407)]
```

Out[31]:
```
[('lcar', 0.8797328472137451),
 ('boxcar', 0.8650722503662109),
 ('ccar', 0.8415129780769348)]
```

В [32]:
```
import string
from tqdm import tqdm_notebook
from tqdm import notebook
from pymorphy2 import MorphAnalyzer
from stop_words import get_stop_words
import annoy
```

B [33]:
```
assert True

#Small preprocess of the answers

question = None
written = False

with open("prepared_text.txt", "w") as fout:
    with open("text.txt", "r", encoding='utf-8') as fin:
        for line in notebook.tqdm(fin):
            if line.startswith("---"):
                written = False
                continue
            if not written and question is not None:
                fout.write(question.replace("\t", " ").strip() + "\t" + line.replace("\t", " "))
                written = True
                question = None
                continue
            if not written:
                question = line.strip()
                continue
```

HBox(children=(HTML(value=''), FloatProgress(value=1.0, bar_style='info', layout=Layout(width='20px'), max=1.0…

B [34]:
```
# %%time

# prepared_text = open('prepared_text.txt','r')
# print(*prepared_text)
```

B [35]:
```
def preprocess_txt(line):
    spls = "".join(i for i in line.strip() if i not in exclude).split()
    spls = [morpher.parse(i.lower())[0].normal_form for i in spls]
    spls = [i for i in spls if i not in sw and i != ""]
    return spls
```

B [36]:
```
assert True

# Preprocess for models fitting

sentences = []

morpher = MorphAnalyzer()
sw = set(get_stop_words("ru"))
exclude = set(string.punctuation)
c = 0

with open("text.txt", "r", encoding='utf-8') as fin:
    for line in notebook.tqdm(fin):
        spls = preprocess_txt(line)
        sentences.append(spls)
        c += 1
        if c > 100000:
            break
```

HBox(children=(HTML(value=''), FloatProgress(value=1.0, bar_style='info', layout=Layout(width='20px'), max=1.0…

B [37]:
```
sentences = [i for i in sentences if len(i) > 2]
```

B [38]:
```
sentences[0]
```

Out[38]:
```
['школотый',
 'поверь',
 'самый',
 'd',
 'общество',
 'профилировать',
 'предмет',
 'тип']
```

2.1 На основе word2vec реализовать метод поиска ближайших твитов

2.2 На основе fasttext реализовать метод поиска ближайших твитов

B [39]:
```
modelW2V = Word2Vec(sentences=sentences, vector_size=300, window=5, min_count=4)
# indexW2V = build_index(modelW2V)
```

B [40]:
```
modelFT = FastText(sentences=sentences, vector_size=300, min_count=4, window=5, workers=8)
# indexFT = build_index(modelFT)
```

B [41]:
```
w2v_index = annoy.AnnoyIndex(300 ,'angular')
ft_index = annoy.AnnoyIndex(300 ,'angular')

index_map = {}
counter = 0

with open("prepared_text.txt", "r") as f:
    for line in tqdm_notebook(f):
        n_w2v = 0
        n_ft = 0
        spls = line.split("\t")
        index_map[counter] = spls[1]
        question = preprocess_txt(spls[0])

        vector_w2v = np.zeros(300)
        vector_ft = np.zeros(300)
        for word in question:
            if word in modelW2V.wv:
                vector_w2v += modelW2V.wv[word]
                n_w2v += 1
            if word in modelFT.wv:
                vector_ft += modelFT.wv[word]
                n_ft += 1
        if n_w2v > 0:
            vector_w2v = vector_w2v / n_w2v
        if n_ft > 0:
            vector_ft = vector_ft / n_ft
        w2v_index.add_item(counter, vector_w2v)
        ft_index.add_item(counter, vector_ft)

        counter += 1

        if counter > 100000:
            break

w2v_index.build(10)
ft_index.build(10)
```

```
<ipython-input-41-e490301d4d3e>:8: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for line in tqdm_notebook(f):
```

HBox(children=(HTML(value=''), FloatProgress(value=1.0, bar_style='info', layout=Layout(width='20px'), max=1.0…

Out[41]:
```
True
```

B [42]:
```
def get_response(question, index, model, index_map):
    question = preprocess_txt(question)
    vector = np.zeros(300)
    norm = 0
    for word in question:
        if word in model.wv:
            vector += model.wv[word]
            norm += 1
    if norm > 0:
        vector = vector / norm
    answers = index.get_nns_by_vector(vector, 5, )
    return [index_map[i] for i in answers]
```

B [43]:
```
TEXT = "хоть я и школота, но поверь, у нас то же самое общество профилирующий предмет типа"
```

B [44]:
```
get_response(TEXT, w2v_index, modelW2V, index_map)
```

Out[44]:
```
['"Да, все-таки он немного похож на него. Но мой мальчик все равно лучше:D"\n',
 '"с утра романтик, засевший глубоко среди извилин мозга, получил суровым кирпичом действительности по голове) и вас с добры"\n',
 '"Надеялась, что я добавлю ее в друзья,лохушка :D http://t.co/36k6ADyASA"\n', (http://t.co/36k6ADyASA"\n',)
 '"Кто круче - 1,2,3,4 ?) #Suri http://t.co/jZ0INfN4tx"\n', (http://t.co/jZ0INfN4tx"\n',)
 '"И вот я сижу .. А он передо мной ..в трусах , а я засыпаю :D"\n']
```

B [45]:
```
get_response(TEXT, ft_index, modelFT, index_map)
```

Out[45]:
```
['"Да, все-таки он немного похож на него. Но мой мальчик все равно лучше:D"\n',
 '"Кто круче - 1,2,3,4 ?) #Suri http://t.co/jZ0INfN4tx"\n', (http://t.co/jZ0INfN4tx"\n',)
 '"И вот я сижу .. А он передо мной ..в трусах , а я засыпаю :D"\n',
 '"время двенадцати нету, а я уже дома:)\n',
 '"с утра романтик, засевший глубоко среди извилин мозга, получил суровым кирпичом действительности по голове) и вас с добры"\n']
```

Вывод

- Сравнивались два метода word2vec и fasttext.
- Первые ответы на вопрос адекватны как для word2vec и fasttext
- Возможно FastText работает немного лучше.