Список работ

Бартеньев О. В.

Программирование моделей текста на Python

Содержание

Введение

При обработке текстов решается большое число задач. Вот некоторые из них [1, 2]:
- предварительная обработка (подготовка) текста;
- выделение составных частей речи, например, после_того_как, кроме_того;
- оценка сложности текста;
- разметка текста по частям речи;
- разметка текста по морфологическим признакам;
- деление слов на морфемы;
- выделение основы слова (стемминг);
- приведение слова к базовой форме (лемматизация);
- исправление ошибок;
- деление текста на предложения (sentence splitting) (возникает при генерации текстов);
- составление из независимых предложений осмысленного текста (sentence fusion);
- расстановка знаков препинания;
- распознавание именованных сущностей (имен собственных, названий географических объектов);
- разрешение лексической многозначности, снятие омонимии (указание на используемый омоним);
- построение синтаксического дерева предложения;
- определение словесных ударений;
- информационный поиск (information retrieval);

- анализ тональности текста (sentiment analysis);
- классификация текстов (документов);
- извлечение данных (знаний) из текстов (information extraction);
- выявление в предложении связанных слов и отношений между словами;
- анализ семантической эквивалентности двух фраз (paraphrase/semantic equivalence analysis);
- анализ семантического сходства текстов (semantic textual similarity – STS);
- определение эквивалентности двух вопросов (question pairs) (*пример* разных вопросов: "What are natural numbers?" – "What is a least natural number?");
– распознавание, является ли смысл одного текста вытекающим из другого текста (recognizing textual entailment);
- разрешение анафоры (anaphora resolution), то есть выяснение, к чему относится местоимение в тексте;
- машинный перевод (machine translation);
- ответы на вопросы (question answering);
- диалоговые модели (как вариант, чат-боты);
- автоматическое реферирование и аннотирование (text summarization);
- изложение текста другими словами (пересказ текста);
- порождение текста;
- понимание прочитанного текста (reading comprehension);
- воспроизведение текста, синтез речи по тексту.

Решение этих задач выполняется с применением моделей текста, предусматривающих представление фрагментов текста, чаще всего слов, в виде вещественных векторов. Модель текста строится на основании достаточно больших корпусов, содержащих либо случайные, например, взятые из Википедии, либо специально подобранные тексты, например, статьи по группе тем.
В работе рассматриваются известные модели текста. В случае непредобученной модели программы, выполняющие формирование корпуса на основе исходного текста и получение моделей текста. Также приводятся примеры употребления моделей в задаче классификации документов.
Источником документов в приводимых примерах является набор новостных данных Би-би-си [3] (далее обозначается как BBCD), содержащий тексты (документы) следующих категорий (после имени указано число документов данной категории): бизнес / 510, развлечения / 386, политика / 417, спорт / 511, техника / 401.
Все приводимые программы написаны на Python.

1. Формирование обучающего и проверочного множеств

BBCD хранит документы в текстовых файлах с кодировкой UTF-8. В каждом файле один документ. Файлы разнесены по папкам с именами, указывающими на класс (категорию) документа: business, entertainment, politics, sport и tech.

Пример документа из папки sport:

Isinbayeva claims new world best
Pole vaulter Yelena Isinbayeva broke her own indoor world record by clearing 4.89 metres in Lievin on Saturday.
It was the Russian's 12th world record of her career and came just a few days after she cleared 4.88m at the Norwich Union Grand Prix in Birmingham. The Olympic champion went on to attempt 5.05m at the meeting on France but failed to clear that height. In the men's 60m, former Olympic 100m champion Maurice Greene could only finish second to Leonard Scott. It was Greene's second consecutive defeat at the hands of his fellow American, who also won in Birmingham last week. "I ran my race perfectly," said Scott, who won in 6.46secs, his best time indoors. "I am happy even if I know that Maurice is a long way from being at his peak at the start of the season."

В процессе загрузки набора формируется корпус, сохраняемый в файле b_x.txt и разбиваемый на обучающее и проверочное множества и сохраняемые соответственно в файлах b_x_t.txt и b_x_v.txt.
Каждая строка файлов хранит один документ.
При формировании корпуса текст приводится в нижний регистр и из него удаляются все символы, отличающиеся от букв английского алфавита. Такие преобразования позволяют снизить размер словаря корпуса, без ущерба решаемой задачи классификации документов.
Одновременно формируются файлы b_y.txt, b_y_t.txt и b_y_v.txt с метками документов (метка – это номер класса документа) из файлов b_y.txt, b_x_t.txt и b_x_v.txt.
Каждая строка *k* в файле b_y.txt (b_y_t.txt, b_y_v.txt) хранит метку документа в строке *k* файла b_x.txt (b_x_t.txt, b_x_v.txt).
Вдобавок создается словарь корпуса, сохраняемый в отсортированном файле b_dict.txt, и файл b_in_cls.txt с номерами классов и числом документов в них:

0 510
1 386
2 417
3 511
4 401

Описанные действия выполняет следующий код:

```
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # <class 'list'>
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
# Сохранение списка в текстовый файл
def add_to_txt_f(lst, fn):
    print('Создан файл', fn, 'с числом строк', len(lst))
    with open(fn, 'w', encoding = 'utf-8') as f:
        for val in lst: f.write((val + '\n') if val.find('\n') == -1 else val)
# Пополнение словарей dict_txt и dict_in_cls
def add_in_dicts(cls, dict_in_cls, doc, dict_txt):
```

```
    in_cls = dict_in_cls.get(cls)
    if in_cls is None:
        dict_in_cls[cls] = 1
    else:
        dict_in_cls[cls] += 1
    lst_t = doc.split()
    for w in lst_t: dict_txt[w] = 1 # Словарь корпуса
    doc = ''
    for w in lst_t: doc += (w + ' ')
    doc = doc.rstrip()
    return doc
# Обработка строки английского текста
def preprocess_en(s):
    s = s.lower()
    # Оставляем только английские строчные буквы, остальное заменяем пробелами
    s = re.sub('[^a-z]', ' ', s)
    # Заменяем одиночные буквы на пробелы
    s = re.sub(r'\b[a-z]\b', ' ', s)
    s = re.sub(' +', ' ', s) # Заменяем несколько пробелов одним
    return s.strip() # Удаляем начальный и конечные пробелы
#
import os, time, re
from sklearn.model_selection import train_test_split
pre_ds = 'b_' # '4_', если BACD
fn_x = pre_ds + 'x.txt'
fn_y = pre_ds + 'y.txt'
fn_d = pre_ds + 'dict.txt'
fn_in_cls = pre_ds + 'in_cls.txt' # Размеры классов
fn_xt = pre_ds + 'x_t.txt'
fn_yt = pre_ds + 'y_t.txt'
fn_xv = pre_ds + 'x_v.txt'
fn_yv = pre_ds + 'y_v.txt'
num_classes = 5 # Число классов
k_split = 0.2 # Доля проверочного множества
pth = 'bbc' # Папка с документами BBCD
#pth = 'e_docs/data' # Папка с документами BACD
t0 = time.time()
print('Подготовка данных')
dict_txt = {} # Словарь корпуса
dict_in_cls = {} # Словарь классов: {номер класса, число документов в классе}
x_trn, y_trn = [], [] # Списки для документов и меток корпуса
i0 = 0
def one_data_set(pth, i0, x_trn_vl, y_trn_vl):
    lst_dir = os.listdir(pth)
    lst_dir = [pth + '/' + dr for dr in lst_dir if dr.find('.') == -1]
    cls = -1
    for dr in lst_dir:
        cls += 1
        print('Формирование класса', cls)
        lst_fn = os.listdir(dr)
        lst_fn = [dr + '/' + fn for fn in lst_fn]
        for fn in lst_fn:
            try:
                lst_s = read_txt_f(fn, say = False)
            except:
                print('ERROR:', fn)
                exit()
            d = ''
            for s in lst_s[i0:]:
                if len(s.strip()) == 0: continue
                s = preprocess_en(s)
                d += s + ' '
            d = d.split()
            if len(d) < 10: continue
            doc = ''
            for w in d:
                doc += w + ' '
            doc = doc.rstrip()
            s_cls = str(cls)
            doc = add_in_dicts(s_cls, dict_in_cls, doc, dict_txt)
            x_trn_vl.append(doc)
            y_trn_vl.append(s_cls)
    return x_trn_vl, y_trn_vl
x_trn, y_trn = one_data_set(pth, i0, x_trn, y_trn)
add_to_txt_f(x_trn, fn_x) # Документы корпуса и их метки
add_to_txt_f(y_trn, fn_y)
xt_all, yt_all, xv_all, yv_all = [], [], [], []
# Делим каждый класс по отдельности, поскольку набор данных несбалансирован
for cls in range(num_classes):
    str_cls = str(cls)
    xt = [x for x, y in zip(x_trn, y_trn) if y == str_cls]
    yt = [str_cls] * len(xt)
```

```
    xt, xv, yt, yv = train_test_split(xt, yt, test_size = k_split, shuffle = False)
    xt_all.extend(xt)
    yt_all.extend(yt)
    xv_all.extend(xv)
    yv_all.extend(yv)
add_to_txt_f(xt_all, fn_xt)
add_to_txt_f(yt_all, fn_yt)
add_to_txt_f(xv_all, fn_xv)
add_to_txt_f(yv_all, fn_yv)
lst_dict = list(dict_txt.keys())
lst_dict.sort()
add_to_txt_f(lst_dict, fn_d)
lst_in_cls = [itm[0] + ' ' + str(itm[1]) for itm in dict_in_cls.items()]
add_to_txt_f(lst_in_cls, fn_in_cls)
print('Длительность подготовки данных:', round(time.time() - t0, 2))
```

Заметим, что дополнительно на этом этапе подготовки данных нередко практикуются *лемматизация* и удаление *стоп-слов*.
В качестве *стоп-слов* указываются междометия, частицы, союзы, вводные слова, удаление которых снижает размер корпуса и его словаря, а также нередко влечет и повышение качества ожидаемого результата.
Лемматизация – это приведение словоформы к лемме, или нормальной форме.
В русском языке леммами считаются следующие морфологические формы [4]:
- имя существительное в именительном падеже и единственном числе;
- имя прилагательное в именительном падеже, единственном числе, мужского рода;
- глагол, причастие, деепричастие – глагол несовершенного вида в инфинитиве.

Примеры:

отпотели скованные ночным заморозком лужи талой воды – *отпотеть скованный ночное заморозок лужа талый вода*;
жалобно скрипели оконные ставни - *жалобно скрипеть оконный ставень*.

При работе с русскими текстами переход к леммам осуществляет следующая функция:

```
# Заменяет слова леммами
def to_normal_form(morph, s):
    s2 = s.split() # Список слов предложения s
    s = ''
    for w in s2:
        w = morph.parse(w)[0].normal_form
        s += (' ' + w)
    return s.lstrip()
import pymorphy2
morph = pymorphy2.MorphAnalyzer()
sen = to_normal_form(morph, 'жалобно скрипели оконные ставни')
```

Аналогичным образом находятся леммы и в английских предложениях, но уже с применением иного лемматизатора:

```
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('wordnet') # Загружаем единожды
lemmatizer = WordNetLemmatizer()
sen = 'rocks corpora'
for w in sen.split():
    print(w, '-', lemmatizer.lemmatize(w))
```

Результат:

```
rocks - rock
corpora - corpus
```

2. Статистические характеристики набора данных

Вычисляются следующие характеристики BBCD:
- число предложений;
- число слов;
- число слогов;
- число многосложных слов (слов с числом слогов более 3);
- число букв;
- число символов, отличных от букв.

```
import os, re
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # <class 'list'>
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
pth = 'bbc' # Папка с документами BBCD
vowels = 'aeiouy' # Гласные
```

```
n_sens = n_words = n_syls = n_c_syls = n_lets = n_all_symbs = 0
lst_dir = os.listdir(pth)
lst_dir = [pth + '/' + dr for dr in lst_dir if dr.find('.') == -1]
for dr in lst_dir:
    lst_fn = os.listdir(dr)
    lst_fn = [dr + '/' + fn for fn in lst_fn]
    for fn in lst_fn:
        lst_d = read_txt_f(fn, say = False)
        for d in lst_d:
            d = d.strip()
            if len(d) == 0: continue
            d = d.lower()
            # Всего символов в документе за вычетом пробелов
            n_d_symbs = len(d) - d.count(' ')
            n_all_symbs += n_d_symbs # Всего символов в наборе данных
            for sen in re.split('[\.?!...]{1,} ', d): # Делим документ на предложения
                n_sens += 1 # Число предложений
                # Оставляем только строчные буквы; остальное заменяем пробелами
                sen = re.sub('[^a-z]', ' ', sen)
                sen = re.sub(' +', ' ', sen) # Заменяем несколько пробелов одним
                sen = sen.rstrip()
                # Всего букв в предложении за вычетом пробелов
                n_sen_lets = len(sen) - sen.count(' ')
                # Всего букв в наборе данных
                n_lets += n_sen_lets
                for w in sen.split():
                    n_words += 1 # Число слов в наборе данных
                    n_w_syls = 0 # Число слогов в слове
                    for c in w:
                        if c in vowels:
                            n_w_syls += 1
                    n_syls += n_w_syls # Число слогов в наборе данных
                    if n_w_syls > 3:
                        n_c_syls += 1 # Число многосложных слов в наборе данных
print('Всего предложений:', n_sens)
print('Всего слов:', n_words)
print('Всего слогов:', n_syls)
print('Всего многосложных слов:', n_c_syls)
print('Всего букв:', n_lets)
print('Всего не букв:', n_all_symbs - n_lets)
```

Статистические характеристики BBCD:

Всего предложений: 42'091;
Всего слов: 859'442;
Всего слогов: 1'579'770;
Всего многосложных слов: 71'560;
Всего букв: 3'989'442;
Всего не букв: 185'087.

К ним можно добавить еще три:

Число классов: 5;
Число документов: 2'225;
Размер словаря корпуса: 27'880.

3. Применяемые классификаторы

Данные, подготовленные моделями текста, передаются классификаторам SGD и НС многослойный перцептрон с одним скрытым слоем.
Создание, обучение и тестирование классификаторов обеспечивает следующая процедура, принимающая векторы и метки документов обучающего и проверочного множеств:

```
def classify(x_trn, y_trn, x_vl, y_vl):
    from sklearn.linear_model import SGDClassifier
    doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)
    doc_clf.fit(x_trn, y_trn) # Обучение классификатора
    print('Оценка точности классификации')
    score = doc_clf.score(x_vl, y_vl)
    print('Точность на проверочном множестве:', round(score, 4))
    score = doc_clf.score(x_trn, y_trn)
    print('Точность на обучающем множестве:', round(score, 4))
    #
    epochs = 90 # Число эпох обучения НС
    num_classes = 5 # Число классов
    def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
        from keras.models import Model
        from keras.layers import Input, Dropout, Dense
        import keras.utils as ut
        print('Преобразуем списки в массивы')
        x_trn = np.array(x_trn, dtype = 'float32')
        x_vl = np.array(x_vl, dtype = 'float32')
```

```
    # Переводим метки в one-hot представление
    y_trn = ut.to_categorical(y_trn, num_classes)
    y_vl = ut.to_categorical(y_vl, num_classes)
    inp_shape = (n_attrs_in_doc, )
    print('Формируем модель НС')
    inp = Input(shape = inp_shape, dtype = 'float32')
    x = Dropout(0.3)(inp)
    x = Dense(32, activation = 'relu')(x)
    output = Dense(num_classes, activation = 'softmax')(x)
    model = Model(inp, output)
    model.summary()
    model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
    # Обучаем НС
    model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
          validation_data = (x_vl, y_vl))
    # Оценка модели НС на оценочных данных
    score = model.evaluate(x_vl, y_vl, verbose = 0)
    # Вывод потерь и точности
    print('Потери при тестировании: ', score[0])
    print('Точность при тестировании:', score[1])
# После преобразования: x_trn, x_vl: class 'numpy.ndarray'
n_attrs_in_doc = len(x_vl[0])
print('Число признаков в документе:', n_attrs_in_doc)
nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
```

4. Оценка модели текста

Классификатор вычисляет две следующие оценки:
val_acc – точность классификации на проверочном множестве;
acc – точность классификации на обучающем множестве.
С каждой моделью текста классификаторы запускаются трижды, после чего находятся средние значения *val_acc* и *acc*.
Для оценки модели текста берется наименьшее из двух полученных средних значений.

5. Модели текста и их программирование

5.1. Рассматриваемые модели текста

Рассматриваются следующие модели текстов: вектор кодов слова и его частей слова (КЧС), one-hot, мешок слов (два варианта), LDA, LSA, частотно-классовая модель (ЧКМ), случайный вектор (СВ), word2vec, doc2vec, fasttext, GloVe и модели на архитектуре трансформеров, такие, как GPT-2, BERT и др.
Все модели представляют в виде вектора слово, или документ, или и слово, и документ. Длина вектора – это либо параметр модели, либо равна числу слов в словаре модели, либо равна числу классов (тем) в корпусе, по которому создается модель.
Рассматриваемые модели, кроме КЧС и СВ, отражают либо частотные характеристики слов корпуса, например LDA, либо связи между словами, например word2vec.
Модели текстов после формирования используются в задаче классификации документов BBCD.
В качестве классификаторов поочередно берутся SGDClassifier библиотеки scikit-learn и нейронная сеть (НС) многослойный перцептрон с одним скрытым слоем.

Замечания.
1. В общем случае модели текста оперируют *токенами*– единицами текста: словами, знаками препинания, числами и пр. В корпусе, созданном на основе BBCD, после предварительной обработки остаются только слова. Поэтому понятие *токен* в излагаемом материале употребляется только при описании моделей, например BERT, обладающих собственными токенизаторами – программами, преобразующими франмент текста в последовательность токенов.
2. Во всех случаях вектор документа, когда он не предоставляется моделью текста, формируется в результате усреднения векторов слов документа.

5.2. Вектор кодов слова и его частей

5.2.1. Описание модели

В КЧС слово *word* представляется в виде списка из n ($n \geq 2$) признаков:

word, *word*[-*n*:], ..., *word*[-2:],

где *word*[-*x*:] – последние *x* букв слова.
Если $n = 1$, то берется только слово.
Части слова дополняются символом '_', что позволяет различить часть слова от слова с таким же написанием. Например, *лед_* в слове *след* не совпадет со словом *лед*.

Пример.

word = 'корова'; $n = 4$. Строковая модель слова: 'корова', 'рова_', 'ова_', 'ва_'.
word = 'след'; $n = 4$. Строковая модель слова: 'след', '', 'лед_', 'ед_'.

Далее на основе этого представления создается модель слова, в которой признаки (слово и его части) заменяются своими числовыми кодами.
Для замены признаков своими кодами составляется единый словарь признаков – слов и их двух, трех, ..., n последних букв.
В качестве числового кода признака берется его номер в этом словаре.
Далее числовые признаки нормируются: код признака делится на размер словаря признаков.

Пример.

Слово *word* = 'корова'; *n* = 3.
Строковая модель слова: ('корова', 'ова_', 'ва_').
Числовая модель слова: [54758, 81020, 73216].
Нормализованная числовая модель слова: [0.5306, 0.78508, 0.70945].

Модель документа получается в результате замены каждого его слова нормализованной числовой моделью слова.

Пример.

```
# Корпус из двух документов
corp = [
   'Корову свою не продам никому –',
   'Такая скотина нужна самому!']
# После обработки
corp = [
   ('корову', 'свою', 'не', 'продам', 'никому'),
   ('такая', 'скотина', 'нужна', 'самому')]
n_atrs = 2 # Число признаков
suff = '_' # Символ, добавляемые в конец части слова
dict_a = {} # Словарь признаков
for doc in corp:
   for w in doc:
      dict_a[w] = 1
      dict_a[w[-2:] + suff] = 1
N = len(dict_a) # Размер словаря признаков
# Нумерация и нормализация числовых значений признаков
v = 0
for a in dict_a.keys():
   v += 1
   dict_a[a] = v / N
lst_c_doc = [] # Числовая модель первого документа
for w in corp[0]:
   lst_c_doc.extend([dict_a[w], dict_a[w[-2:] + suff]])
print(lst_c_doc)
# [0.0625, 0.125, 0.1875, 0.25, 0.3125, 0.375, 0.4375, 0.5, 0.5625, 0.625]
```

Основной недостаток модели в том, что она не отражает ни частотных характеристик корпуса, ни отношений между его словами.

5.2.2. Подготовка данных и результаты

В задаче классификации документов с применением КЧС формируется список lst_corp_codes, содержащие числовые представления документов.
Номера их классов загружаются в список и lst_cls.
Формирование списков выполняется в результате выполнения следующих действий:

1. Загрузить в список lst_dict словарь корпуса текста.
2. Создать основании списка lst_dict словарь dict_a, с ключом "слово " или "часть слова" и значением "номер (части) слова в словаре".
3. Загрузить документы корпуса и их метки соответственно в списки lst_corp и lst_cls.
4. lst_corp_codes = [] # Модель корпуса
5. Для каждого документа doc из lst_corp:
   doc_codes = [0]*n_attrs # Модель документа – массив из *n_attrs* нулей
   nw = 0 # Число слов в документе
   Для каждого слова w из doc:
      nw = nw + 1
      Создать массив w_codes из n_attrs числовых признаков слова w
      (значения числовых признаков берутся из словаря dict_a).
      doc_codes = doc_codes + w_codes.
   doc_codes = doc_codes / nw
   Добавить doc_codes в lst_corp_codes.

Таким образом, каждый документ заменяется вектором из *n_attrs* числами, где *n_attrs* – число признаков в модели слова.
Названные выше действия реализует следующий код:

```
import numpy as np
print('Модель слова в виде кодов слова и его частей')
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
   with open(fn, 'r', encoding = encoding) as f:
      lst = f.readlines() # class 'list'
   if no_n:
      lst = [x.replace('\n', '') for x in lst]
   if to_int:
      lst = [int(x) for x in lst] # При чтении меток
   if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
   return lst
# Формирование из списка слов словаря с элементами {слово:код слова}
def make_dict_from_list(lst, k0):
   dict_x = {}
   k = k0
   for x in lst:
      k += 1
```

```
            if dict_x.get(x) is None:
                dict_x.update({x : k})
        return dict_x
    # Пополняет словарь признаков по списку строковых признаков
    def update_dict_a(lst_a, dict_a, suff = '_'):
        lst_a = list(set(lst_a))
        for a in lst_a:
            a += suff
            if dict_a.get(a) is None:
                dict_a.update({a : 1})
        return dict_a
    # Формирует список числовых признаков по списку строковых признаков
    def make_lst_c(lst_a, dict_a, suff = '_'):
        lst_c = []
        for a in lst_a:
            a += suff
            c = dict_a.get(a)
            c = [0] if c is None else [c]
            lst_c.extend(c)
        return lst_c
    def add_ends(dict_a, n_attrs):
        lst_a = []
        for w in dict_a.keys():
            for n in range(2, n_attrs + 1):
                lst_a.append(w[-n:])
        return update_dict_a(lst_a, dict_a)
    # Формируем список числовых признаков слова w
    def find_ends(w, dict_a, n_attrs):
        lst_a = []
        for n in range(2, n_attrs + 1):
            lst_a.append(w[-n:])
        return make_lst_c(lst_a, dict_a)
    def read_corp():
        path = ''
        fn_xt, fn_yt, fn_d = 'b_x_t.txt', 'b_y_t.txt', 'b_dict.txt'
        fn_xv, fn_yv = 'b_x_v.txt', 'b_y_v.txt'
        x_trn = read_txt_f(path + fn_xt)
        y_trn = read_txt_f(path + fn_yt, to_int = True)
        x_vl = read_txt_f(path + fn_xv)
        y_vl = read_txt_f(path + fn_yv, to_int = True)
        # Список из слов словаря корпуса
        lst_dict = read_txt_f(fn_d)
        return x_trn, y_trn, x_vl, y_vl, lst_dict
    def train_test_data(x_trn, x_vl, lst_dict):
        n_attrs = 4 # Число признаков в модели слова
        print('Число признаков в слове:', n_attrs)
        # Формируем начальный словарь атрибутов
        dict_a = make_dict_from_list(lst_dict, 0)
        # Добавляем в словарь части слов и получаем полный словарь атрибутов
        dict_a = add_ends(dict_a, n_attrs)
        lst_dict_a = list(dict_a.keys())
        lst_dict_a.sort()
        N = len(lst_dict_a) # Размер словаря атрибутов
        dict_a = make_dict_from_list(lst_dict_a, 0)
        # Приводим значения признаков к диапазону (0, 1] (нормализация)
        for a in dict_a.keys(): dict_a[a] /= N
        def mk_x_trn_vl_codes(x_trn_vl, n_attrs, dict_a):
            x_trn_vl_codes = []
            # Замена слов на числовые признаки слова и его частей
            nc_max = 0
            for doc in x_trn_vl:
                doc = doc.split()
                nw = 0
                lst_codes = np.zeros(n_attrs)
                for w in doc:
                    cw = dict_a.get(w)
                    if cw is not None:
                        lst_c = [cw]
                        lst_c.extend(find_ends(w, dict_a, n_attrs))
                        nw += 1
                        lst_codes += np.array(lst_c)
                lst_codes /= nw
                x_trn_vl_codes.append(lst_codes)
            return x_trn_vl_codes
        x_trn = mk_x_trn_vl_codes(x_trn, n_attrs, dict_a)
        x_vl = mk_x_trn_vl_codes(x_vl, n_attrs, dict_a)
        return x_trn, x_vl
    def classify(x_trn, y_trn, x_vl, y_vl):
        from sklearn.linear_model import SGDClassifier
        doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)
        doc_clf.fit(x_trn, y_trn) # Обучение классификатора
        print('Оценка точности классификации')
```

```
    score = doc_clf.score(x_vl, y_vl) # class 'numpy.float64'
    print('Точность на проверочном множестве:', round(score, 4))
    score = doc_clf.score(x_trn, y_trn) # class 'numpy.float64'
    print('Точность на обучающем множестве:', round(score, 4))
    #
    epochs = 90 # Число эпох обучения НС
    num_classes = 5 # Число классов
    def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
        from keras.models import Model
        from keras.layers import Input, Dropout, Dense
        import keras.utils as ut
        print('Преобразуем списки в массивы')
        x_trn = np.array(x_trn, dtype = 'float32')
        x_vl = np.array(x_vl, dtype = 'float32')
        # Переводим метки в one-hot представление
        y_trn = ut.to_categorical(y_trn, num_classes)
        y_vl = ut.to_categorical(y_vl, num_classes)
        inp_shape = (n_attrs_in_doc, )
        print('Формируем модель НС')
        inp = Input(shape = inp_shape, dtype = 'float32')
        x = Dropout(0.3)(inp)
        x = Dense(32, activation = 'relu')(x)
        output = Dense(num_classes, activation = 'softmax')(x)
        model = Model(inp, output)
        model.summary()
        model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        # Обучаем НС
        model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
                validation_data = (x_vl, y_vl))
        # Оценка модели НС на оценочных данных
        score = model.evaluate(x_vl, y_vl, verbose = 0)
        # Вывод потерь и точности
        print('Потери при тестировании: ', score[0])
        print('Точность при тестировании:', score[1])
    # После преобразования: x_trn, x_vl: class 'numpy.ndarray'
    n_attrs_in_doc = len(x_vl[0])
    print('Число признаков в документе:', n_attrs_in_doc)
    nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
#
x_trn, y_trn, x_vl, y_vl, lst_dict = read_corp()
x_trn, x_vl = train_test_data(x_trn, x_vl, lst_dict)
#
# Полученные векторы документов передаются классификаторам – SGD и НС
# Классификация выполняется трижды
# Результаты (точность классификации) усредняются
for k in range(3):
    print('Номер попытки:', k + 1)
    classify(x_trn, y_trn, x_vl, y_vl)
```

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
1	24.78	24.59	22.99	24.65
2	30.36	30.67	31.92	23.47
3	32.14	32.70	31.03	23.58

5.3. One-hot модель слова

5.3.1. Описание модели

В этой модели слово заменяется вектором размера N, где N – число слов в словаре корпуса.
Если номер слова в словаре равен k, то в векторе, замещающем слово, компонент k - 1 равен 1, а остальные компоненты вектора равны нулю.
Чтобы получить one-hot представление документа, нужно заменить каждое его слово на соответствующий one-hot вектор.

Пример.

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np
# Корпус из 4-х документов
corp = [
    'Заяц в лес бежал по лугу,',
    'Я из лесу шел домой, –',
    'Бедный заяц с перепугу',
    'Так и сел передо мной!']
# После обработки
corp = [
    ('заяц', 'в', 'лес', 'бежал', 'по', 'лугу'),
    ('я', 'из', 'лесу', 'шел', 'домой'),
    ('бедный', 'заяц', 'с', 'перепугу'),
    ('так', 'и', 'сел', 'передо', 'мной')]
```

```
encoder = OneHotEncoder(sparse = False)
# Список слов корпуса
corp_words = []
for doc in corp:
    corp_words.extend([w for w in doc])
# Массив слов корпуса для encoder.fit
arr_words = np.array(corp_words).reshape(len(corp_words), 1)
one_hot = encoder.fit(arr_words)
# Имена признаков (словарь)
feature_names = one_hot.get_feature_names()
# arr_one_hot_words – массив формы (len(corp_words), len(feature_names))
# Массив one-hot представлений слов документа
arr_one_hot_words = one_hot.transform(arr_words)
# Вывод one-hot представлений слов первого документа
print(arr_one_hot_words[:len(corp[0]), :])
[[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] # 'заяц'
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] # 'в'
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] # 'лес'
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] # 'бежал'
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.] # 'по'
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]] # 'лугу'
```

Можно сразу получить векторы всех слов:

```
arr_one_hot_words = encoder.fit_transform(arr_words) # class 'numpy.ndarray'
```

One-hot модель первого документа:

```
nw = 0
one_hot_doc = np.zeros(len(feature_names))
for one_hot_word in arr_one_hot_words[:len(corp[0]), :]:
    nw += 1
    one_hot_doc  += one_hot_word
one_hot_doc /= nw
[0. 0.167 0.167 0. 0.16 0. 0. 0.167 0. 0.167 0. 0. 0. 0.167 0. 0. 0. 0. 0.]
```

Очевидные недостатки модели – это ее большой размер, игнорирование связей между словами и неразличение омонимов.

5.3.2. Подготовка данных и результаты

Подготовка данных для обучения и тестирования классификаторов выполняется по файлам, содержащим обработанные документы и список слов словаря корпуса.
Данные формирует и передает классификаторам следующий код:

```
import numpy as np
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
def read_corp():
    path = ''
    fn_xt, fn_yt, fn_d = 'b_x_t.txt', 'b_y_t.txt', 'b_dict.txt'
    fn_xv, fn_yv = 'b_x_v.txt', 'b_y_v.txt'
    x_trn = read_txt_f(path + fn_xt)
    y_trn = read_txt_f(path + fn_yt, to_int = True)
    x_vl = read_txt_f(path + fn_xv)
    y_vl = read_txt_f(path + fn_yv, to_int = True)
    # Список из слов словаря корпуса
    lst_dict = read_txt_f(fn_d)
    return x_trn, y_trn, x_vl, y_vl, lst_dict
def train_test_one_hot(x_trn, x_vl, lst_dict):
    from sklearn.preprocessing import OneHotEncoder
    print('Модель one-hot')
    len_dict = len(lst_dict)
    encoder = OneHotEncoder(sparse = False)
    arr_words = np.array(lst_dict).reshape(len_dict, 1)
    arr_one_hot_words = encoder.fit_transform(arr_words)
    # Словарь корпуса: {слово:one-hot представление слова}
    dict_corp = {}
    for w, c in zip(lst_dict, arr_one_hot_words):
        dict_corp[w] = c
    arr_words = arr_one_hot_words = '' # Для сокращения издержек памяти
    def make_trn_vl_1(x_trn_vl):
        x_trn_vl_one_hot = []
        # Замена слов на one-hot представления
        nc_max = 0
        for doc in x_trn_vl:
```

```
            doc = doc.split()
            nw = nv = 0
            cw_sum = np.zeros(len_dict)
            for w in doc:
                c = dict_corp.get(w)
                if c is not None:
                    nw += 1
                    cw_sum += c
            x_trn_vl_one_hot.append(cw_sum / nw)
        return x_trn_vl_one_hot
    x_trn = make_trn_vl_1(x_trn)
    x_vl = make_trn_vl_1(x_vl)
    return x_trn, x_vl
def classify(x_trn, y_trn, x_vl, y_vl):
    from sklearn.linear_model import SGDClassifier
    doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)
    doc_clf.fit(x_trn, y_trn) # Обучение классификатора
    print('Оценка точности классификации')
    score = doc_clf.score(x_vl, y_vl)
    print('Точность на проверочном множестве:', round(score, 4))
    score = doc_clf.score(x_trn, y_trn)
    print('Точность на обучающем множестве:', round(score, 4))
    #
    epochs = 90 # Число эпох обучения НС
    num_classes = 5 # Число классов
    def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
        from keras.models import Model
        from keras.layers import Input, Dropout, Dense
        import keras.utils as ut
        print('Преобразуем списки в массивы')
        x_trn = np.array(x_trn, dtype = 'float32')
        x_vl = np.array(x_vl, dtype = 'float32')
        # Переводим метки в one-hot представление
        y_trn = ut.to_categorical(y_trn, num_classes)
        y_vl = ut.to_categorical(y_vl, num_classes)
        inp_shape = (n_attrs_in_doc, )
        print('Формируем модель НС')
        inp = Input(shape = inp_shape, dtype = 'float32')
        x = Dropout(0.3)(inp)
        x = Dense(32, activation = 'relu')(x)
        output = Dense(num_classes, activation = 'softmax')(x)
        model = Model(inp, output)
        model.summary()
        model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        # Обучаем НС
        model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
                  validation_data = (x_vl, y_vl))
        # Оценка модели НС на оценочных данных
        score = model.evaluate(x_vl, y_vl, verbose = 0)
        # Вывод потерь и точности
        print('Потери при тестировании: ', score[0])
        print('Точность при тестировании:', score[1])
    # После преобразования: x_trn, x_vl: class 'numpy.ndarray'
    n_attrs_in_doc = len(x_vl[0])
    print('Число признаков в документе:', n_attrs_in_doc)
    nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
#
x_trn, y_trn, x_vl, y_vl, lst_dict = read_corp()
x_trn, x_vl = train_test_one_hot(x_trn, x_vl, lst_dict)
#
# Полученные векторы документов передаются классификаторам – SGD и НС
# Классификация выполняется трижды
# Результаты (точность классификации) усредняются
for k in range(3):
    print('Номер попытки:', k + 1)
    classify(x_trn, y_trn, x_vl, y_vl)
```

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
1	86.61	86.10	99.33	99.94
2	91.74	87.90	99.11	99.94
3	87.95	88.91	99.11	100.0

5.4. Вектор присутствия слов

5.4.1. Описание модели

Документ *d* представляется вектором *x*, который формируется по словарю dict_w с элементами {*word*:*i*}, где *i* – номер *word* в dict_w), следующим образом:

1. Взять вектор *x* из *N* нулей, где *N* – размер словаря dict_w.
2. Для Каждого *word* Из *d*:
   i = dict_w[word]
   x[i] = x[i] + 1

Таким образом, вектор, представляющий документ, содержит в позиции с индексом, равным номеру слова в словаре, число присутствий слова в документе.
Другое название модели – *мешок слов* [5].
Вектор, представляющий документ, можно сделать бинарным:

Если x[i] == 0: x[i] = x[i] + 1

Принцип построения небинарного вектора иллюстрирует следующий *пример*:

```
from sklearn.feature_extraction.text import CountVectorizer
corp = [
   'Заяц в лес бежал по лугу,',
   'Я из лесу шел домой, –',
   'Бедный заяц с перепугу (заяц)',
   'Так и сел передо мной!']
vectorizer = CountVectorizer(token_pattern = '\w+') # binary = True
x = vectorizer.fit_transform(corp) # x – class 'scipy.sparse.csr.csr_matrix'
feature_names = vectorizer.get_feature_names() # class 'list'
arr_x = x.toarray() # arr_x – class 'numpy.ndarray'
print(feature_names)
print(arr_x)
```

Признаки (*feature_names*):

['бедный', 'бежал', 'в', 'домой', 'заяц', 'и', 'из', 'лес', 'лесу', 'лугу', 'мной', 'передо', 'перепугу', 'по', 'с', 'сел', 'так', 'шел', 'я']

Модель корпуса (arr_x):

```
[[0 1 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0] # 'заяц в лес бежал по лугу'
[0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1] # 'я из лесу шел домой'
[1 0 0 0 2 0 0 0 0 0 0 0 1 0 1 0 0 0 0] # 'бедный заяц с перепугу заяц'
[0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 0 0]] # 'так и сел передо мной'
```

Замечание. CountVectorizer без *token_pattern* = '\w+' исключит из списка признаков токены единичной длиной – *в, и, с, я*.

Если задать

vectorizer = CountVectorizer(token_pattern = '\w+', binary = True),

то векторы, представляющие документы, будут бинарными, то есть содержать только нули и единицы.
Положительная черта такого кодирования документов – это инвариантность к его длине: любой документ отображается вектором, длина которого равна числу слов в словаре. Недостаток заключается в игнорировании связей между словами корпуса.
Больший эффект, например, при обучении классификатора можно получить, если выполнить нормализацию данных, возвращаемых методом CountVectorizer, применив TfidfTransformer:

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.pipeline import Pipeline
corp = [
   'Заяц в лес бежал по лугу,',
   'Я из лесу шел домой, —',
   'Бедный заяц с перепугу (заяц)',
   'Так и сел передо мной!']
t_vec = Pipeline([('count', CountVectorizer()), ('tfid', TfidfTransformer())]).fit(corp)
arr = t_vec.transform(corp).toarray()
print(arr)
[[0. 0.421 0.421 0. 0.332 0. 0. 0.421 0. 0.421 0. 0. 0. 0.421 0. 0. 0. 0. 0.]
[0. 0. 0. 0.447 0. 0. 0.447 0. 0.447 0. 0. 0. 0. 0. 0. 0. 0. 0.447 0.447]
[0.426 0. 0. 0. 0.673 0. 0. 0. 0. 0. 0. 0.426 0. 0.426 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0.447 0. 0. 0. 0. 0.447 0.447 0. 0. 0. 0.447 0.447 0. 0.]]
```

TfidfTransformer использует метод кодирования TF-IDF – Term Frequency-Inverse Document Frequency (частота слова в документе-обратная частота документов, содержащих слово), в котором значение признака – это нормированная величина показателя *tf_idf*, вычисляемого по следующей формуле [6]:

$tf\_idf = \mathrm{tf}(w_t, d) * \mathrm{idf}(w_t)$,

где
$\mathrm{tf}(w_t, d)$ – число появлений слова w_t в документе d;
$\mathrm{idf}(w_t) = \log(n / \mathrm{df}(w_t)) + 1$, если *smooth_idf* = False;
$\mathrm{idf}(w_t) = \log((1 + n) / (1 + \mathrm{df}(w_t))) + 1$, если *smooth_idf* = True
(*smooth_idf* – параметр метода TfidfTransformer);
n – число документов в корпусе;
$\mathrm{df}(w_t)$ – число документов, в которых встречается слово w_t.
Такой способ кодирования позволяет повысить значимость редко встречаемых в документе слов (значимость слова тем выше, чем больше значение *tf_idf*).
Приведенный в предыдущем примере конвейер Pipeline можно записать с одним объединяющим конвертором:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
# TfidfVectorizer = CountVectorizer + TfidfTransformer
t_vec = TfidfVectorizer(analyzer = 'word', max_features = None)
t_vec.fit(corp)
print(t_vec.transform(corp).toarray())
```

Данные, получаемые на выходе CountVectorizer и TfidfVectorizer, имеют тип scipy.sparse.csr.csr_matrix, то есть хранятся в разреженном csr-представлении.

Пример.

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[1, 0, 0, 1, 0, 0], [0, 0, 2, 0, 0, 1], [0, 0, 0, 2, 0, 0]])
#[[1 0 0 1 0 0]
# [0 0 2 0 0 1]
# [0 0 0 2 0 0]]
# Показатель разреженности массива
sparsity = 1.0 - np.count_nonzero(arr) / arr.size
print(round(sparsity, 2)) # 0.72
#
# Преобразование массива в csr-матрицу
csr_mat = csr_matrix(arr)
#  (0, 0)    1
#  (0, 3)    1
#  (1, 2)    2
#  (1, 5)    1
#  (2, 3)    2
# Преобразование csr-матрицы в плотный массив:
arr = csr_mat.todense()
# или:
arr = csr_mat.toarray()
```

5.4.2. Подготовка данных и результаты

В задаче классификации документов при использовании рассматриваемой модели на вход SGDClassifier подаются матрицы в разреженном csr-представлении, подготовленные CountVectorizer и TfidfVectorizer:

```
import numpy as np
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
def read_corp():
    path = ''
    fn_xt, fn_yt, fn_d = 'b_x_t.txt', 'b_y_t.txt', 'b_dict.txt'
    fn_xv, fn_yv = 'b_x_v.txt', 'b_y_v.txt'
    x_trn = read_txt_f(path + fn_xt)
    y_trn = read_txt_f(path + fn_yt, to_int = True)
    x_vl = read_txt_f(path + fn_xv)
    y_vl = read_txt_f(path + fn_yv, to_int = True)
    # Список из слов словаря корпуса
    lst_dict = read_txt_f(fn_d)
    return x_trn, y_trn, x_vl, y_vl, lst_dict
def classify(x_trn, y_trn, x_vl, y_vl):
    from sklearn.linear_model import SGDClassifier
    doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)
    doc_clf.fit(x_trn, y_trn) # Обучение классификатора
    print('Оценка точности классификации')
    score = doc_clf.score(x_vl, y_vl)
    print('Точность на проверочном множестве:', round(score, 4))
    score = doc_clf.score(x_trn, y_trn)
    print('Точность на обучающем множестве:', round(score, 4))
    #
    epochs = 90 # Число эпох обучения НС
    num_classes = 5 # Число классов
    def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
        from keras.models import Model
        from keras.layers import Input, Dropout, Dense
        import keras.utils as ut
        # Переводим метки в one-hot представление
        y_trn = ut.to_categorical(y_trn, num_classes)
        y_vl = ut.to_categorical(y_vl, num_classes)
        inp_shape = (n_attrs_in_doc, )
        print('Формируем модель НС')
        inp = Input(shape = inp_shape, dtype = 'float32')
        x = Dropout(0.3)(inp)
        x = Dense(32, activation = 'relu')(x)
```

```
        output = Dense(num_classes, activation = 'softmax')(x)
        model = Model(inp, output)
        model.summary()
        model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        # Обучаем НС
        model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
                validation_data = (x_vl, y_vl))
        # Оценка модели НС на оценочных данных
        score = model.evaluate(x_vl, y_vl, verbose = 0)
        # Вывод потерь и точности
        print('Потери при тестировании: ', score[0])
        print('Точность при тестировании:', score[1])
    # После преобразования: x_trn, x_vl: class 'numpy.ndarray'
    n_attrs_in_doc = len(x_vl[0])
    print('Число признаков в документе:', n_attrs_in_doc)
    nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
for clf in ['CV', 'TF']:
    print('Модель:', clf)
    x_trn, y_trn, x_vl, y_vl, lst_dict = read_corp()
    if clf == 'CV':
        from sklearn.feature_extraction.text import CountVectorizer
        vec = CountVectorizer(token_pattern = '\w+', binary = False)
    elif clf == 'TF':
        from sklearn.feature_extraction.text import TfidfVectorizer
        vec = TfidfVectorizer(analyzer = 'word', binary = False)
    len_trn = len(x_trn)
    x_trn.extend(x_vl) # Объединяем x_trn и x_vl и получаем полный корпус
    x_trn = vec.fit_transform(x_trn)
    x_vl = x_trn[len_trn:]
    x_trn = x_trn[:len_trn]
    # Преобразуем разреженные матрицы в массивы
    x_trn = np.float32(x_trn.toarray())
    x_vl = np.float32(x_vl.toarray())
    #
    # Полученные векторы документов передаются классификаторам – SGD и НС
    # Классификация выполняется трижды
    # Результаты (точность классификации) усредняются
    for k in range(3):
        print('Номер попытки:', k + 1)
        classify(x_trn, y_trn, x_vl, y_vl)
```

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
CountVectorizer (CV)				
1	96.88	100.0	99.11	100.0
2	97.99	100.0	98.66	100.0
3	96.43	100.0	98.88	100.0
TfidfVectorizer (Tf-idf)				
1	98.88	100.0	98.66	100.0
2	99.11	100.0	98.88	100.0
3	98.66	100.0	98.88	100.0

5.5. Латентное размещение Дирихле

5.5.1. Описание модели

Латентное размещение Дирихле (LDA) [7] – это один из методов тематического моделирования, в котором определяются распределения *слово – тема* и *документ – тема*. В качестве априорного берется распределение Дирихле. Число тем указывается в качестве параметра метода.
При описании алгоритма LDA [7] используются следующие понятия:
- слово – элемент словаря с индексами {1, ..., V}.
Для представления слова используется модель one-hot: слово с индексом v представляется вектором w размера V, в котором $w[v] = 1$, а прочие компоненты – 0;
- *документ* – последовательность N слов $\boldsymbol{d} = (w_1, ..., w_N)$;
- *корпус* – коллекция M документов $D = (\boldsymbol{d}_1, ..., \boldsymbol{d}_M)$.
Решается задача поиска вероятностной модели корпуса – поиска распределений *слово – тема* и *документ – тема*.
Алгоритм LDA основывается на предположении, что каждый документ корпуса относится к одной из скрытых (неизвестных) тем, и каждая тема характеризуется некоторым распределением слов. Последовательность реализации LDA:

Распределение Пуассона, можно взять иное
1. Выбрать $N \sim$ Poisson(ξ).
2. Выбрать $\theta \sim$ Dir(α). # Распределение Дирихле
3. Для каждого документа d из D:
   # Мультиноминальное распределение
   Выбрать тему $z_n \sim$ Multinomial(θ).
   Выбрать с вероятностью $p(w_n \mid z_n, \beta)$ слово w_n, где $p(w_n \mid z_n, \beta)$ – мультиномиальная условная вероятность принадлежности слова w_n теме z_n.

В [7] сделано несколько упрощающих допущений. Во-первых, размерность k распределения Дирихле и, следовательно,

число тем предполагаются известными и фиксированными. Во-вторых, вероятности слов параметризуются $k \times V$ матрицей β, где $\beta_{ij} = p(w_j = 1 \mid z_i = 1)$, которая рассматривается как фиксированная величина, подлежащая определению.
Далее в [7] приводятся формулы для распределений *слово – тема*, *документ – тема* и алгоритм получения результата – вероятностей принадлежности слов и документов k темам.

Пример получения LDA методами библиотеки scikit-learn; число тем 3 (n_components = 3).

```
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer
# Корпус из 4-х документов
corp = ['Заяц в лес бежал по лугу,',
   'Я из лесу шел домой, –',
   'Бедный заяц с перепугу (заяц)',
   'Так и сел передо мной!']
vec = CountVectorizer(token_pattern = '\w+', binary = False)
x = vec.fit_transform(corp) # x – class 'scipy.sparse.csr.csr_matrix'
lda = LatentDirichletAllocation(n_components = 3, random_state = 0)
lda.fit(x) # Получаем LDA
print('Вероятности "слово - тема"')
print(lda.components_) # Массив формы (n_components, размер_словаря)
# [[0.33380389 1.33167202 ... 1.33144111]
# [0.33462881 0.33449496 ... 0.33473411]
# [1.3315673 0.33383302 ... 0.33382478]]
x = lda.transform(x) # Вероятности "документ – тема"
print('Вероятности "документ - тема"')
for d in x:
   print(d)
[0.90036875 0.04861569 0.05101557]
[0.88730686 0.05672797 0.05596517]
[0.05723641 0.05640735 0.88635625]
[0.05591795 0.05665323 0.88742883]
```

Чтобы получить LDA, методу fit из LatentDirichletAllocation нужно передать векторизованный корпус в виде *мешка слов*, подготовку которого выполняет метод fit_transform из CountVectorizer.

5.5.2. Подготовка данных и результаты

Результатом подготовки данных является бинарный файл, хранящий векторы с вероятностями *документ-тема*.
Число тем задается равным числу классов в корпусе. Каждый вектор отвечает одному документу корпуса, и является, таким образом, векторным представлением документа.
Формирование векторных представлений документов корпуса на основе LDA обеспечивает следующий код:

```
import numpy as np
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
   with open(fn, 'r', encoding = encoding) as f:
      lst = f.readlines() # class 'list'
   if no_n:
      lst = [x.replace('\n', '') for x in lst]
   if to_int:
      lst = [int(x) for x in lst] # При чтении меток
   if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
   return lst
fn_xt, fn_xv, fn_wv = 'b_x_t.txt', 'b_x_v.txt', 'b_LDA.bin'
x_trn = read_txt_f(fn_xt)
x_vl = read_txt_f(fn_xv)
x_trn.extend(x_vl) # Объединяем x_trn и x_vl и получаем полный корпус
num_classes = 5 # Число классов (тем)
print('Создание LDA-векторов. Число тем:', num_classes)
vec = CountVectorizer(token_pattern = '\w+', binary = False)
x_trn = vec.fit_transform(x_trn)
lda = LatentDirichletAllocation(n_components = num_classes, random_state = 0)
x_trn = lda.fit_transform(x_trn) # sklearn.decomposition._lda.LatentDirichletAllocation
fn = open(fn_wv, 'wb')
fn.write(np.float32(x_trn))
fn.close()
```

На этапе классификации после модели загрузки в массив меняется его форма и затем устанавливается соответствие *номер темы – номер класса*, такое, при котором наблюдается максимальная точность классификации. После этого в списках с метками последние заменяются на соответствующие номера тем.
Поскольку LDA-векторы содержат вероятности *документ – тема* (*класс*), то номера предсказанного класса – это индекс наибольшего элемента LDA-вектора.
Документ классифицирован верно, если номер предсказанного класса совпадает с номером метки документа.

```
import numpy as np
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
   with open(fn, 'r', encoding = encoding) as f:
      lst = f.readlines() # class 'list'
   if no_n:
      lst = [x.replace('\n', '') for x in lst]
```

```
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
# Вычисление точности по вектору частоты
# (вектору вероятности принадлежности документа классу)
def one_eval(knd, x, y):
    n_true = 0
    for vec, cls in zip(x, y):
        if cls == vec.argmax(): n_true += 1
    print('Точность на ' + knd + ' множестве', round(n_true / len(y) * 100, 2))
fn_yt, fn_yv, fn_wv = 'b_y_t.txt', 'b_y_v.txt', 'b_LDA.bin'
y_trn = read_txt_f(fn_yt, to_int = True)
y_vl = read_txt_f(fn_yv, to_int = True)
num_classes = 5 # Число классов (тем)
with open(fn_wv, 'rb') as fn:
    x_trn = np.fromfile(fn , dtype = 'float32')
len_trn = len(y_trn)
x_trn.shape = (len_trn + len(y_vl), num_classes) # Все векторы
# Нужно LDA-темам найти соответствующие номера классов документов
# Алгоритм:
# Взять документы класса i
# Найти точность классификаций для всех тем LDA и поставить классу i в соответствие
# тему с наибольшей точностью классификации
y = y_trn.copy()
y.extend(y_vl)
lst_n = []
for c in range(num_classes):
    lst = [0]*num_classes
    for cls, vec in zip(y, x_trn):
        if c == cls:
            c2 = vec.argmax()
            lst[c2] += 1
    lst_n.append([c, lst])
cls_numb = [0]*num_classes # Список соответствий класс - тема
for c in range(num_classes):
    lst = lst_n[c][1]
    m = np.array(lst).argmax()
    cls_numb[c] = m
##for c in range(num_classes): print(lst_n[c])
##print(cls_numb)
# Уточняем cls_numb вручную:
cls_numb = [1, 2, 4, 3, 0] # bbc
def make_new_y(y_trn_vl, cls_numb):
    y = [cls_numb[cls] for cls in y_trn_vl]
    return y
y_trn = make_new_y(y_trn, cls_numb)
x_vl = x_trn[len_trn:] # Делим векторы на проверочные и обучающие
x_trn = x_trn[:len_trn]
y_vl = make_new_y(y_vl, cls_numb)
one_eval('проверочном', x_vl, y_vl) # Точность классификации
one_eval('обучающем', x_trn, y_trn)
```

Результат:

```
Точность на проверочном множестве 68.53%
Точность на обучающем множестве 71.86%
```

Если же задать, например, 768 тем и LDA-векторы документов подать на вход классификаторов то получим в случае SGD:

Точность на проверочном множестве 81.70% Точность на обучающем множестве 92.35%

и в случае HC:

Точность на проверочном множестве 86.38% Точность на обучающем множестве 91.67%

5.6. Латентно-семантический анализ

Опираясь на латентно-семантический анализ (LSA) [8] (он же LSI – латентно-семантическое индексирование), можно выполнять тематическое моделирование, выделяя в документе темы и слова в этих темах.
Вероятностная разновидность метода (pLSA), подобно LDA, находит распределения *документ – тема* и *слово – тема*, которые, в частности, можно употребить для решения задачи классификации.

Пример получения LSA-векторов:

```
def treat_corp(corp):
    from gensim import corpora
    # Теперь каждый документ - это список слов
    docs = [d.split() for d in corp]
    # Словарь корпуса
    dct = corpora.Dictionary(docs) # len(dct) = len(lst_dict)
    # Получаем матрицу документ - слово (мешок слов)
    doc_term_matrix = [dct.doc2bow(doc) for doc in docs]
```

```
    return dct, doc_term_matrix
#
num_classes = 5 # Число классов (тем)
x_trn, y_trn, x_vl, y_vl, _ = read_corp()
len_trn = len(x_trn)
x_trn.extend(x_vl)
dct, doc_term_matrix = treat_corp(x_trn)
from gensim.models import LsiModel
# Обучение модели
lsa_model = LsiModel(doc_term_matrix, num_topics = num_classes, id2word = dct)
for t_words in lsa_model.print_topics(num_topics = num_classes, num_words = 5):
    print(t_words)
# 5 слов в каждой теме
# (0, '0.708*"the" + 0.340*"to" + 0.275*"of" + 0.254*"and" + 0.223*"in"')
# (1, '0.488*"the" + -0.375*"to" + -0.291*"is" + -0.247*"that" + -0.226*"it"')
# (2, '0.543*"he" + 0.301*"to" + 0.287*"mr" + 0.263*"said" + 0.244*"his"')
# (3, '0.392*"to" + 0.250*"said" + -0.246*"it" + -0.234*"and" + 0.231*"of"')
# (4, '-0.662*"in" + 0.356*"the" + -0.308*"and" + 0.172*"mr" + 0.112*"that"')
# class 'gensim.interfaces.TransformedCorpus'
lsa_vecs = lsa_model[doc_term_matrix] # Получаем векторы [(тема, координата), ...]
# Можно получить один вектор:
lsa_vec0 = lsa_model[doc_term_matrix[0]]
print(x_trn[0])
print(doc_term_matrix[0])
print(lsa_vecs[0])
print(lsa_vec0) # То же, что и lsa_vecs[0]
# Документ: ad sales boost time warner... on the value of that stake
# Doc2bow-модель документа: [(0, 2), (1, 1), (2, 1), ..., (208, 4)]
# Lsi-вектор: [(0, 34.07), (1, -3.36), (2, 0.82), (3, -6.05), (4, -5.45)]
# dct[208] - это year; year встречается в первом документе 4 раза: (208, 4)
x_trn = [[v[1] for v in vecs] for vecs in lsa_vecs]
# Векторы проверочного и обучающего множеств
x_vl = x_trn[len_trn:]
x_trn = x_trn[:len_trn]
```

LSA-модель применяется, например, для кластеризации документов, поиска данных, выработки рекомендаций в результате анализа профилей пользователей.

5.7. Частотно-классовая модель

5.7.1. Описание модели

В ЧКМ, подобно LDA, вычисляются вероятности *слово – класс*. В отличие от LDA, вычисления выполняются на обучающем множестве с использованием сведений о классе документа: для каждого слова из словаря корпуса формируется вектор p длины C (C – число классов), в котором $p_c = n_c / n_d$, где n_c – число присутствий слова в документах класса c, а n_d – число присутствий слова в документах всех классов.
На основе векторов слов формируются векторы документов обоих множеств: обучающего и проверочного. Номер класса документа так же, как и при работе с LDA, определяется по индексу максимального значения в векторе документа.

5.7.2. Подготовка данных и результаты

При подготовке данных формируется словарь, в котором для каждого слова указан вектор p, элемент $p[c]$ которого – суть число присутствий слова в классе c набора данных.
Далее элементы вектора делятся на число присутствий слова во всех классах и интерпретируются как вероятности попадания слова в классы.
Подготовку данных и вычисление точности классификации реализует следующий код:

```
import numpy as np
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
def read_corp():
    path = ''
    fn_xt, fn_yt, fn_d = 'b_x_t.txt', 'b_y_t.txt', 'b_dict.txt'
    fn_xv, fn_yv = 'b_x_v.txt', 'b_y_v.txt'
    x_trn = read_txt_f(path + fn_xt)
    y_trn = read_txt_f(path + fn_yt, to_int = True)
    x_vl = read_txt_f(path + fn_xv)
    y_vl = read_txt_f(path + fn_yv, to_int = True)
    # Список из слов словаря корпуса
    lst_dict = read_txt_f(fn_d)
    return x_trn, y_trn, x_vl, y_vl, lst_dict
def make_dict_f_words(x_trn, y_trn, num_classes):
    dict_all_words = {}
    dict_cls_words = {}
```

```
    for cls in range(num_classes):
        dict_cls_words[cls] = {}
    for d, cls in zip(x_trn, y_trn):
        d = d.split()
        d_cls = dict_cls_words[cls]
        for w in d:
            if dict_all_words.get(w) is None:
                dict_all_words[w] = 1
            else:
                dict_all_words[w] += 1
            if d_cls.get(w) is None:
                d_cls[w] = 1
            else:
                d_cls[w] += 1
    dict_f_words = dict_all_words.copy()
    for w, k in dict_f_words.items():
        dict_f_words[w] = np.zeros(num_classes)
    for cls in range(num_classes):
        d_cls = dict_cls_words[cls]
        for w, f in d_cls.items():
            f = f / dict_all_words[w]
            dict_f_words[w][cls] = f
    return dict_f_words
def make_trn_vl_15(lst, dict_f_words):
    x_trn_vl = []
    for d in lst:
        d = d.split()
        nw = nv = 0
        v = np.zeros(num_classes)
        for w in d:
            f = dict_f_words.get(w)
            if f is not None:
                v += f
                nw += 1
        x_trn_vl.append(v / nw)
    return x_trn_vl
# Вычисление точности по вектору частоты
# (вектору вероятности принадлежности документа классу)
def one_eval(knd, x, y):
    n_true = 0
    for vec, cls in zip(x, y):
        if cls == vec.argmax(): n_true += 1
    print('Точность на ' + knd + ' множестве', round(n_true / len(y) * 100, 2))
num_classes = 5 # Число классов
x_trn, y_trn, x_vl, y_vl, _ = read_corp()
dict_f_words = make_dict_f_words(x_trn, y_trn, num_classes)
x_trn = make_trn_vl_15(x_trn, dict_f_words)
x_vl = make_trn_vl_15(x_vl, dict_f_words)
one_eval('проверочном', x_vl, y_vl)
one_eval('обучающем', x_trn, y_trn)
```

Результат:

```
Точность на проверочном множестве 95.09%
Точность на обучающем множестве 98.42%
```

5.8. Случайный вектор

5.8.1. Описание модели

В CB для каждого слова из словаря генерируется на основе равномерного распределения вектор размера *size* (*size* = 150). Затем формируется модель документа, в которой каждое слово заменяется соответствующим сгенерированным для него вектором, а вектор документа создается в результате усреднения векторов слов документа.

5.8.2. Подготовка данных и результаты

Подготовку данных и передачу их классификаторам выполняет следующий код:

```
import numpy as np
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
def read_corp():
    path = ''
    fn_xt, fn_yt, fn_d = 'b_x_t.txt', 'b_y_t.txt', 'b_dict.txt'
```

```
    fn_xv, fn_yv = 'b_x_v.txt', 'b_y_v.txt'
    x_trn = read_txt_f(path + fn_xt)
    y_trn = read_txt_f(path + fn_yt, to_int = True)
    x_vl = read_txt_f(path + fn_xv)
    y_vl = read_txt_f(path + fn_yv, to_int = True)
    # Список из слов словаря корпуса
    lst_dict = read_txt_f(fn_d)
    return x_trn, y_trn, x_vl, y_vl, lst_dict
# Формирование из списка слов словаря с элементами {слово:код слова}
def make_dict_from_list(lst, k0):
    dict_x = {}
    k = k0
    for x in lst:
        k += 1
        if dict_x.get(x) is None:
            dict_x.update({x : k})
    return dict_x
def rand_vecs(x_trn, x_vl, lst_dict, size = 150):
    print('Случайно генерируемые векторы слов')
    print('Размер вектора:', size)
    dict_a = make_dict_from_list(lst_dict, 0) # Формируем словарь
    for w in dict_a.keys():
        dict_a[w] = list(np.random.uniform(-1, 1, [size]))
    def make_trn_vl_5(x_trn_vl):
        x_trn_vl_codes = [] # Список векторных представлений документов
        for doc in x_trn_vl:
            doc = doc.split()
            cw_sum = np.zeros(size)
            nw = 0 # nw - число векторов, добавленных к cw_sum
            for w in doc:
                if dict_a.get(w) is not None:
                    cw = dict_a[w]
                    cw_sum += cw
                    nw += 1
            x_trn_vl_codes.append(cw_sum / nw)
        return x_trn_vl_codes
    x_trn = make_trn_vl_5(x_trn)
    x_vl = make_trn_vl_5(x_vl)
    return x_trn, x_vl
def classify(x_trn, y_trn, x_vl, y_vl):
    from sklearn.linear_model import SGDClassifier
    doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)
    doc_clf.fit(x_trn, y_trn) # Обучение классификатора
    print('Оценка точности классификации')
    score = doc_clf.score(x_vl, y_vl) # class 'numpy.float64'
    print('Точность на проверочном множестве:', round(score, 4))
    score = doc_clf.score(x_trn, y_trn) # class 'numpy.float64'
    print('Точность на обучающем множестве:', round(score, 4))
    #
    epochs = 90 # Число эпох обучения НС
    num_classes = 5 # Число классов
    def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
        from keras.models import Model
        from keras.layers import Input, Dropout, Dense
        import keras.utils as ut
        print('Преобразуем списки в массивы')
        x_trn = np.array(x_trn, dtype = 'float32')
        x_vl = np.array(x_vl, dtype = 'float32')
        # Переводим метки в one-hot представление
        y_trn = ut.to_categorical(y_trn, num_classes)
        y_vl = ut.to_categorical(y_vl, num_classes)
        inp_shape = (n_attrs_in_doc, )
        print('Формируем модель НС')
        inp = Input(shape = inp_shape, dtype = 'float32')
        x = Dropout(0.3)(inp)
        x = Dense(32, activation = 'relu')(x)
        output = Dense(num_classes, activation = 'softmax')(x)
        model = Model(inp, output)
        model.summary()
        model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        # Обучаем НС
        model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
                  validation_data = (x_vl, y_vl))
        # Оценка модели НС на оценочных данных
        score = model.evaluate(x_vl, y_vl, verbose = 0)
        # Вывод потерь и точности
        print('Потери при тестировании: ', score[0])
        print('Точность при тестировании:', score[1])
    # После преобразования: x_trn, x_vl: class 'numpy.ndarray'
    n_attrs_in_doc = len(x_vl[0])
    print('Число признаков в документе:', n_attrs_in_doc)
    nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
```

```
x_trn, y_trn, x_vl, y_vl, lst_dict = read_corp()
x_trn, x_vl = rand_vecs(x_trn, x_vl, lst_dict, size = 150)
#
# Полученные векторы документов передаются классификаторам – SGD и НС
# Классификация выполняется трижды
# Результаты (точность классификации) усредняются
for k in range(3):
    print('Номер попытки:', k + 1)
    classify(x_trn, y_trn, x_vl, y_vl)
```

Результаты:

Номер попытки	SGD		НС	
	val_acc	*acc*	*val_acc*	*acc*
size = 768				
1	94.54	99.94	95.98	97.69
2	90.40	99.83	96.21	97.07
3	93.08	99.77	95.98	96.00
size = 1024				
1	96.65	99.94	98.21	98.37
2	97.32	99.89	98.44	98.65
3	96.88	99.94	98.44	98.31

5.9. Word2vec

5.9.1. Описание модели

Word2vec (word to vector), наряду с такими моделями, как GloVe (global vectors for word representation) и fasttext, учитывает связи между словами в предложениях корпуса. В word2vec реализованы две модели – это CBOW (непрерывный мешок слов) и Skip-Gram (n-грамма с пропуском слова) [9]. В обеих моделях слова представляются в виде вещественных векторов заданного размера. Компоненты вектора размера *n* интерпретируются как координаты слова в *n*-мерном пространстве представления слов.
Векторы получаются в результате обучения НС либо прогнозировать слово по контексту (CBOW), либо контекст по слову (Skip-Gram).
Формально word2vec максимизируется правдоподобие корпуса, заданного последовательностью слов w_1, ..., w_T:

$$\text{CBOW:}\sum_{t=1}^{T}\sum_{c\in C_t}\log\bigl(p(w_t|c)\bigr),$$

$$\text{Skip-Gram:}\sum_{t=1}^{T}\sum_{c\in C_t}\log\bigl(p(c|w_t)\bigr),$$

где $p(w_t|c)$ и $p(c|w_t)$ – соответственно условные вероятности предсказания слова w_t по контексту c и контекста по слову.
C_t – множество контекстов слова w_t.
Каждый контекст – это слова, окружающие w_t. Размер контекста является одним из параметров модели.
НС обучается на некотором корпусе. При его изменении word2vec-модель следует создать заново.
Задача прогнозирования сводится к задаче классификации. В случае CBOW на вход НС подается контекст слова *word* – последовательность заданного числа правых и левых соседей слова *word*. Каждое слово контекста представлено своим числовым кодом.
На выходе имеем прогноз НС – вектор размера *num_words* + 1, где *num_words* – число слов в словаре корпуса.
Модель НС, созданная средствами Keras, может иметь указанные на рис. 1 слои; их описание дано в табл. 1.

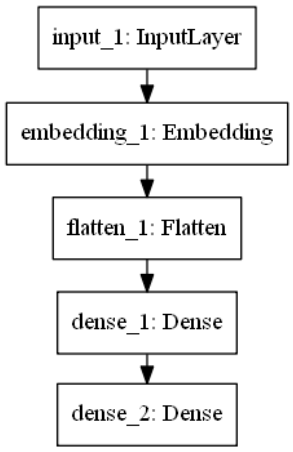

Рис. 1. Слои НС при реализации CBOW

Таблица 1. Описание слоев НС при реализации CBOW

Слой (тип)	Форма выхода	Число параметров
input_1 (InputLayer)	(None, 4)	0
embedding_1 (Embedding)	(None, 4, 50)	905100
flatten_1 (Flatten)	(None, 200)	0
dense_1 (Dense)	(None, 50)	10050
dense_2 (Dense)	(None, 18102)	923202

Искомые векторы являются весами слоя dense_2 и получаются после обучения НС следующим образом:

```
from keras import backend as K
def get_wei(model, nL):
    L = model.layers[nL]
    wei_d = L.weights
    n_w = len(wei_d)
    bias = None
    if n_w == 0:
        wei = None
    else:
        wei = K.eval(wei_d[0])
        if n_w > 1:
            bias = wei_d[1]
            bias = K.eval(bias)
    return wei, bias
wei, _ = get_wei(model, 4) # model – модель НС
```

Код формирования модели НС:

```
from keras.models import Model
from keras.layers import Embedding
from keras.layers import Input, Dense, Flatten
from keras import initializers
import keras.losses as ls
def create_model(sq, num_words, size):
    w_init = initializers.RandomNormal()
    inp = Input(shape = (sq, ), dtype = 'int32')
    x = Embedding(num_words, output_dim = size, input_length = sq,
                embeddings_initializer = w_init, trainable = True)(inp)
    x = Flatten()(x)
    x = Dense(size, activation = 'linear', use_bias = True)(x)
    output = Dense(num_words, activation = ' softmax', use_bias = True)(x)]
    model = Model(inp, output)
    model.summary()
    return model
model = create_model(sq, num_words, size)
model.compile(optimizer = 'adam', loss = ls.binary_crossentropy, metrics = ['accuracy'])
```

В приведенном коде:
sq – размер контекста (*sq* = 4);
size – размерность пространства представления (длина вектора, представляющего слово, *size* = 50);
num_words – размер словаря корпуса текста + одно слово (*num_words* = 18102).
Структура НС при получении модели Skip-Gram при *sq* = 4 показана на рис. 2.

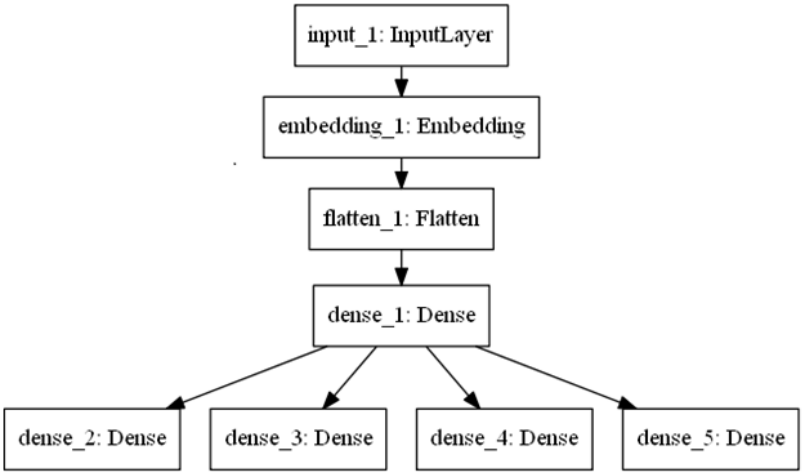

Рис. 2. Структура НС при реализации Skip-Gram

Код создания модели НС отличается только определением ее выходных слоев, задаваемых списком *outputs*:

```
outputs = []
for k in range(sq):
    outputs.append(Dense(num_words, activation = 'linear')(x))
model = Model(inp, outputs = outputs)
```

Заметим, что при обучении НС применяются как положительные, так и отрицательные примеры [9].
Так, в случае Skip-Gram положительные примеры – это найденные в корпусе контексты слова, поданного на вход НС, а отрицательные – это контексты, сформированные из случайно выбранных слов словаря корпуса.
Использование приведенных на рис. 1 и 2 НС связано с большими временными затратами и, кроме того, требует предварительной обработки корпуса текста, в частности, для устранения перекосов обучения, возникающих из-за наличия часто употребляемых слов [9].
Быстрое и качественное построение word2vec-модели можно выполнить средствами библиотеки Gensim [gensim], подавая на вход построителя следующие параметры:
data – список абзацев корпуса следующего вида:

```
[['мой', 'дядя', 'самых', 'честных', 'правил'], ..., ];
```

sg = 0 – используем модель CBOW (по умолчанию); *sg* = 1 – используем модель Skip-Gram (по умолчанию *sg* = 0);
window – максимальное расстояние между текущим словом и словами около него;
min_count – слово должно встречаться в корпусе текста минимум *min_count* раз, чтобы модель его учитывала;
n_iter - число итераций.

```
import multiprocessing
from gensim.models import Word2Vec
from gensim.models.word2vec import LineSentence
data = LineSentence(<имя текстового файла с корпусом>)
workers = multiprocessing.cpu_count()
model = Word2Vec(data, size = size, window = window, min_count = min_cnt,
                    sg = sg, workers = workers, iter = n_iter) # iter = 5 (по умолчанию)
```

В обученной модели близкие по значению слова позиционируются рядом в пространстве представления слов.
Методы word2vec позволяют выполнять различные операции со словами, а точнее с представляющими их векторами, например, вычислять расстояния между ними или получать слова, наиболее близкие к заданному.
Модель имеет недостатки. В частности, она не учитывает порядок слов и не различает омонимы. Так, глагол и существительное *пила* (ср. *девочка пила воду* и *пила стояла за печкой*) будут представлены в модели одним вектором.

5.9.2. Некоторые методы и свойства word2vec-модели

Создадим word2vec-модель по корпусу текста, загрузив его из файла corp.txt, и запишем ее в файл w2v.model.

```
def make_word2vec(sg, size, window, min_cnt, n_iter, fn_c, fn_wv, wv_train = False):
    import multiprocessing
    from gensim.models import Word2Vec
    from gensim.models.word2vec import LineSentence
    print('Создание word2vec-модели по файлу', fn_c)
    print('Читаем файл', fn_c)
    # Загружаем записи файла и преобразуем их в списки:
    # получим
    # ['жалобно', 'скрипели', 'оконные', 'ставни']
    # вместо
    # 'жалобно скрипели оконные ставни'
    data = LineSentence(fn_c) # class 'gensim.models.word2vec.LineSentence'
    #for x in data: print(x)
    workers = multiprocessing.cpu_count()
    print('Создаем word2vec-модель: \n sg =', sg, '\n size =', size,
          '\n min_cnt =', min_cnt, '\n window =', window, '\n n_iter =', n_iter,
          '\n multiprocessing.cpu_count =', workers)
    start_time = time.time() # Время начала создания word2vec-модели
    if wv_train:
        print('Модель создается за три этапа: wv_train =', wv_train)
        model = Word2Vec(size = size, window = window, min_count = min_cnt,
                         sg = sg, workers = workers, iter = n_iter)
        model.build_vocab(data)
        cnt = model.corpus_count ## Число строк в корпусе текста
        model.train(data, total_examples = cnt, epochs = n_iter, report_delay = 1)
    else:
        print('Модель создается за один этап: wv_train =', wv_train)
        model = Word2Vec(data, size = size, window = window, min_count = min_cnt,
                         sg = sg, workers = workers, iter = n_iter) # iter = 5 (по умолчанию)
    print('Время создания word2vec-модели:', round(time.time() - start_time, 0))
    print('word2vec-модель записана в файл', fn_wv)
    model.save(fn_wv)
    return model
#
sg, size, window, min_cnt, n_iter  = 0, 50, 3, 2, 30
make_word2vec(sg, size, window, min_cnt, n_iter, 'corp.txt', 'w2v.model')
```

Примеры использования методов и свойств word2vec приведены в следующем коде:

```
from gensim.models import Word2Vec
# Некотрые методы и свойства word2vec
# Загрузка ранее созданной word2vec-модели
model = Word2Vec.load('w2v.model')
# Число строк (предложений) в корпусе
print(model.corpus_count) # 12'197
# Всего слов в корпусе
print(model.corpus_total_words) # 110'339
# wv – индексированные векторы слов
wv = model.wv
# Словарь модели
vocab = wv.vocab
# Число слов в словаре модели
n_words_wv = len(vocab)
print(n_words_wv) # 13'966
#
# Слова для последующего использования
word = 'глаза'
word2 = 'сердце'
word3 = 'хлеб'
#
```

```
# Координаты слова word
print(wv[word])
# или
print(wv.get_vector(word))
#
# Расстояние между словами
print(wv.distance(word, word2)) # 0.44
print(wv.distance(word, word3)) # 0.59
print(wv.distance(word2, word3)) # 0.04
#
# Массив расстояний между словом и словами из последующего кортежа
print(wv.distances(word, (word, word2, word3))) # [0. 0.44 0.59]
# Массив расстояний между словом и всеми прочими словами
print(wv.distances(word)) # [1.17 0.8 0.5 ... 0.53 0.42 0.47]
#
# Список кортежей, содержащих слова, наиболее близко расположенные
# в созданной модели к заданному слову
sims = model.similar_by_word(word)
print(sims)
# word = 'глаза'
# [('лицо', 0.89), ('подсчитывая', 0.87), ('закрыл', 0.86),
#  ('губы', 0.85), ('словно', 0.85), ('голову', 0.83),
#  ('ноги', 0.82), ('увидел', 0.81), ('широко', 0.81), ('сквозь', 0.81)]
#
# Список кортежей, содержащих слова, наиболее близко расположенные
# к заданному вектору
vec = wv.get_vector(word)
sims = model.similar_by_vector(vec)
# word = 'глаза'
print(sims) # [('глаза', 1.), а далее то же, что и выше,
# но без последнего кортежа
#
# Слова, наиболее близкие к вектору -vec
sims = model.similar_by_vector(-vec)
print(sims)
#
# Слова, наиболее близкие к разнице двух слов: дед – отец
m_sim = wv.most_similar(positive = ['дед'], negative = ['отец'])
# или
vec1 = wv.get_vector('дед')
vec2 = wv.get_vector('отец')
m_sims = model.similar_by_vector(vec1 - vec2)
print('дед – отец')
print(m_sim)
#
# Слова, наиболее близкие к сумме двух слов: мать + отец
m_sim = wv.most_similar(positive = ['мать', 'сын'], negative = [])
# или
vec1 = wv.get_vector('мать')
vec2 = wv.get_vector('сын')
m_sims = model.similar_by_vector(vec1 + vec2)
print('мать + отец')
print(m_sim)
#
# Слова, противоположные слову 'война'
m_sim = wv.most_similar(positive = [], negative = ['война'])
print("negative = ['война']")
print(m_sim)
#
# Список слов словаря
words_list = list(vocab.keys())
# Список слов словаря, упорядоченный по числу вхождений слова в корпус
# (сортировка по убыванию)
words_list = wv.index2word
#
print('')
# Список координат слов
vec_list = wv.syn0
# или
vec_list = wv.vectors
#
# Вывод трех случайно выбранных слов и их координат
for k in range(3):
    n = np.random.randint(n_words_wv - 1)
    word = words_list[n]
    vec = vec_list[n]
    print(word)
    print(vec)
#
# Прогноз центрального слова, заданного списком слов контекста
sen = 'поднялся тихо ступая босыми ногами'
sen = sen.split()
```

```
pred_words = model.predict_output_word(
   [sen[0], sen[1], sen[3], sen[4]], topn = 5)
print(pred_words)
```

5.9.3. Подготовка данных и результаты

Подготовка данных для обучения и проверки классификатора выполняется с использованием word2vec-модели, формируемой следующим кодом:

```
from gensim.models import Word2Vec
from gensim.models.word2vec import LineSentence
import multiprocessing
fn_c, fn_wv = 'b_x.txt', 'b_w2v.model'
print('Создание word2vec-модели по файлу', fn_c)
sg, size, window, min_cnt, n_iter = 0, 150, 3, 1, 100
workers = multiprocessing.cpu_count()
data = LineSentence(fn_c) # class 'gensim.models.word2vec.LineSentence'
model = Word2Vec(data, size = size, window = window, min_count = min_cnt,
                 sg = sg, workers = workers, iter = n_iter)
model.save(fn_wv)
```

В задаче классификации документ независимо от числа слов представляется вектором размера *size* (*size* = 150). Вектор документа получается в результате усреднения векторов его слов. Длина вектора слова так же равна *size*. Формирование векторных представлений документов и последующую классификацию обеспечивает следующий код:

```
import numpy as np
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
   with open(fn, 'r', encoding = encoding) as f:
      lst = f.readlines() # class 'list'
   if no_n:
      lst = [x.replace('\n', '') for x in lst]
   if to_int:
      lst = [int(x) for x in lst] # При чтении меток
   if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
   return lst
def read_corp():
   path = ''
   fn_xt, fn_yt, fn_d = 'b_x_t.txt', 'b_y_t.txt', 'b_dict.txt'
   fn_xv, fn_yv = 'b_x_v.txt', 'b_y_v.txt'
   x_trn = read_txt_f(path + fn_xt)
   y_trn = read_txt_f(path + fn_yt, to_int = True)
   x_vl = read_txt_f(path + fn_xv)
   y_vl = read_txt_f(path + fn_yv, to_int = True)
   # Список из слов словаря корпуса
   lst_dict = read_txt_f(fn_d)
   return x_trn, y_trn, x_vl, y_vl, lst_dict
def vocab_from_model(fn_wv):
   from gensim.models import Word2Vec
   model = Word2Vec.load(fn_wv)
   wv = model.wv # wv – индексированные векторы слов
   vocab = wv.vocab
   for w in vocab.keys():
      vec = wv[w]
      return wv, vocab, len(vec) # Модель, словарь модели и длина вектора слова
def make_trn_vl_4(x_trn_vl, wv, vocab, size):
   x_trn_vl_codes = [] # Список векторных представлений документов
   for doc in x_trn_vl:
      doc = doc.split()
      cw_sum = np.zeros(size)
      nv = nw = 0 # nw - число векторов, добавленных к cw_sum или в lst_codes
      for w in doc:
         in_vocab = vocab.get(w)
         if in_vocab is not None:
            cw = wv[w]
            cw_sum += cw
            nw += 1
      x_trn_vl_codes.append(cw_sum / nw)
   return x_trn_vl_codes
def classify(x_trn, y_trn, x_vl, y_vl):
   from sklearn.linear_model import SGDClassifier
   doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)
   doc_clf.fit(x_trn, y_trn) # Обучение классификатора
   print('Оценка точности классификации')
   score = doc_clf.score(x_vl, y_vl)
   print('Точность на проверочном множестве:', round(score, 4))
   score = doc_clf.score(x_trn, y_trn)
   print('Точность на обучающем множестве:', round(score, 4))
   #
   epochs = 90 # Число эпох обучения НС
   num_classes = 5 # Число классов
   def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
```

```
        from keras.models import Model
        from keras.layers import Input, Dropout, Dense
        import keras.utils as ut
        print('Преобразуем списки в массивы')
        x_trn = np.array(x_trn, dtype = 'float32')
        x_vl = np.array(x_vl, dtype = 'float32')
        # Переводим метки в one-hot представление
        y_trn = ut.to_categorical(y_trn, num_classes)
        y_vl = ut.to_categorical(y_vl, num_classes)
        inp_shape = (n_attrs_in_doc, )
        print('Формируем модель НС')
        inp = Input(shape = inp_shape, dtype = 'float32')
        x = Dropout(0.3)(inp)
        x = Dense(32, activation = 'relu')(x)
        output = Dense(num_classes, activation = 'softmax')(x)
        model = Model(inp, output)
        model.summary()
        model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        # Обучаем НС
        model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
                validation_data = (x_vl, y_vl))
        # Оценка модели НС на оценочных данных
        score = model.evaluate(x_vl, y_vl, verbose = 0)
        # Вывод потерь и точности
        print('Потери при тестировании: ', score[0])
        print('Точность при тестировании:', score[1])
    # После преобразования: x_trn, x_vl: class 'numpy.ndarray'
    n_attrs_in_doc = len(x_vl[0])
    print('Число признаков в документе:', n_attrs_in_doc)
    nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
#
x_trn, y_trn, x_vl, y_vl, _ = read_corp()
wv, vocab, size = vocab_from_model('b_w2v.model')
x_trn = make_trn_vl_4(x_trn, wv, vocab, size)
x_vl = make_trn_vl_4(x_vl, wv, vocab, size)
#
# Полученные векторы документов передаются классификаторам – SGD и НС
# Классификация выполняется трижды
# Результаты (точность классификации) усредняются
for k in range(3):
    print('Номер попытки:', k + 1)
    classify(x_trn, y_trn, x_vl, y_vl)
```

Для сравнения взяты также и предобученные векторы размера 100, предоставленные в [52].
Для ускорения процесса классификации по загруженному из [52] файлу (его размер 3,7M) по словарю корпуса BBCD формируется файл с координатами слов корпуса:

```
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
dct = read_txt_f('b_dict.txt')
dict_b = {}
for w in dct: dict_b[w] = 1
in_dict = len(dict_b)
print(in_dict)
f = open('word2vec/model.txt', 'r', encoding = 'utf-8')
f2 = open('word2vec/b_w2v_txt.txt', 'w', encoding = 'utf-8')
n = n2 = 0
while True:
    n += 1
    if n % 50000 == 0: print(n)
    try: # Могут быть проблемы с кодировкой
        s = f.readline()
    except:
        print(n)
        print(s)
        continue
    if not s: break
    p = s.find(' ')
    w = s[:p]
    if dict_b.get(w) is not None:
        n2 += 1
        f2.write(s)
        if n2 == in_dict: break
f.close()
f2.close()
```

По данным файла на этапе классификации формируется словарь векторов слов, который затем используется для формирования векторов, представляющих документы (см. GloVe):

```
fn_x = 'word2vec/b_w2v_txt.txt'
corp = read_txt_f(fn_x)
dict_x = {}
for d in corp:
    d = d.split()
    for w in d:
        dict_x[w] = 1
t0 = time.time()
wv = {} # Словарь векторов слов
with open(fn_wv, 'r', encoding = 'utf-8') as f:
    for vec in f:
        vec = vec.split()
        word = vec[0] # Токен из файла fn_wv
        if dict_x.get(word) is not None:
            vec = np.array(vec[1:], dtype = 'float32')
            wv[word] = vec
print('Размер словаря:', len(wv))
print('Время создания словаря:', round(time.time() - t0, 2))
```

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
***size* = 100, предобученная [52]**				
1	96.43	96.51	94.42	91.11
2	97.54	96.96	94.87	91.22
3	96.21	96.51	94.87	91.28
***size* = 150**				
1	96.65	97.92	97.77	97.86
2	98.66	98.37	97.99	97.81
3	96.65	97.86	97.99	97.86
***size* = 768**				
1	97.32	99.83	98.21	98.59
2	97.52	99.16	98.44	98.26
3	97.32	99.66	98.44	98.37

5.10. Doc2vec

5.10.1. Описание модели

Doc2vec находит не только векторные представления слов, но и документов корпуса. Размеры векторов, формируемых doc2vec для слов и документов, одинаковы (задаются параметром *vector_size*).
Модель doc2vec основывается на модели распределенного представления слов [10], в которой векторы слов получаются в результате обучения НС (классификатора) предсказывать слово по заданному левому контексту (рис. 3).

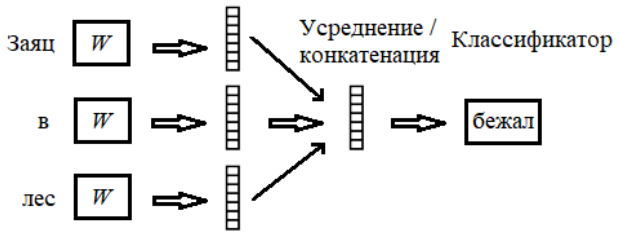

Рис. 3. Схема обучения классификатора в модели распределенного представления слов

В этой модели каждое слово корпуса представляется в виде вектора – столбца матрицы W. Индекс столбца – это номер слова в словаре.
На вход классифицирующего слоя передаются признаки, получаемые в результате усреднения суммы векторов, или конкатенация векторов.
Формально задача записывается следующим образом. Задан корпус в виде последовательности слов w_1, ..., w_T. Необходимо максимизировать правдоподобие корпуса – сумму логарифмов условной вероятности появления слова с номером t после k слов, предшествующих этому слову (на рис. 3 k = 3).

$$\sum_{t=k}^{T-k} \log p(w_t | w_{t-k}, \dots, w_{t-1}), \qquad (5.1)$$

где w_{t-k}, ..., w_{t-1} – левый контекст слова w_t.
Вероятности моделируются функцией softmax

$$\text{softmax}(y) = e^y / \sum_i e^{y_i}$$

Пример:

```
import numpy as np
h = np.array([3, 4, 5], dtype = 'double')
```

```
# Показываем результат с точностью 4 знака
sum_e = sum(np.exp(h)) # np.exp(3) + np.exp(4) + np.exp(5) = 223.0969
sf_h = h / sum_e # [0.01345 0.0179 0.0224]
```

Таким образом, в (5.1)

$$p(w_t|w_{t-k}, \dots, w_{t-1}) = e^{y_{w_t}} / \sum_i e^{y_i},$$

где y_i – ненормализованный логарифм вероятности для каждого слова на выходе классификатора, вычисляемый как

$y = b + Uh\ (w_{t-k}, \dots, w_{t-1}; W)$,

где U, b – параметры выходного слоя классификатора, а h – вектор, получаемый в результате конкатенации или усреднения суммы векторов, извлекаемых из матрицы W.
Число классов равно размеру словаря, которое в случае использования отрицательных примеров увеличивается на 1.
На практике для ускорения вычислений используется иерархический softmax [11].
В качестве структуры в иерархическом softmax может быть использовано бинарное дерево Хаффмана [9], в котором короткие коды назначаются часто используемым словам, в результате ускоряется их поиск и, как следствие, снижается время вычисления softmax.
Так же как и в word2vec, при надлежащем корпусе и правильно организованном обучении классификатора получаемые векторы слов таковы, что расстояние между векторами близких по значения слов меньше, чем для слов с несхожими значениями.

Пример применения средств библиотеки tensorflow для усреднения, конкатенации и вычисления softmax:

```
import tensorflow as tf
a = tf.constant([[1, 2], [4, 5]], dtype = 'float32')
b = tf.constant([[5, 6], [8, 7]], dtype = 'float32')
# На входе слоев Average и Concatenate список тензоров
avg = tf.keras.layers.Average()([a, b])
cnc = tf.keras.layers.Concatenate()([a, b])
sf_avg = tf.keras.activations.softmax(avg)
sf_cnc = tf.keras.activations.softmax(cnc)
sess = tf.Session()
print(sess.run(avg)) # [[3. 4.] [6. 6.]]
print(sess.run(cnc)) # [[1. 2. 5. 6.] [4. 5. 8. 7.]]
# Результат приводится с точностью 4 знака
print(sess.run(sf_avg)) # [[0.2689 0.7311] [0.5 0.5]]
print(sess.run(sf_cnc)) # [[0.0048 0.0132 0.2641 0.7179] [0.0128 0.03467 0.6964 0.2562]]
```

В doc2vec при обучении классификатора дополнительно определяется вектор абзаца [12], причем имеются две реализации: PV-DM – вектор абзаца с распределенной памятью (рис. 4) и PV-DBOW – вектор абзаца с распределенным мешком слов (рис. 5).

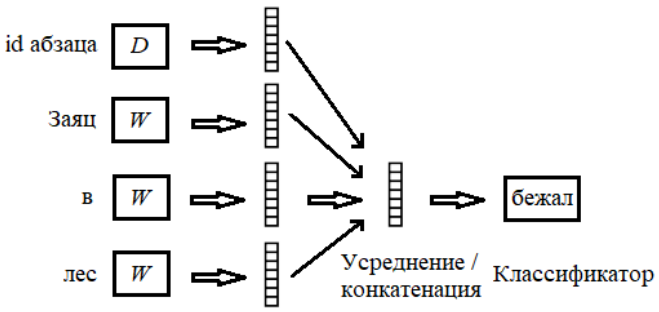

Рис. 4. Doc2vec с PV-DM

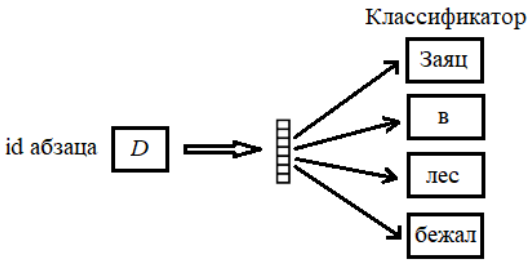

Рис. 5. Doc2vec с PV-DBOW

Вектор абзаца хранит *id*-столбец матрицы *D*.
В реализации PV-DM классификатор обучается предсказывать следующее слово по известным абзацу и левому контексту. И абзац, и слова контекста задаются векторами, постоянно уточняемыми в процессе обучения.
В реализации PV-DBOW классификатор обучается предсказывать по вектору абзаца случайно выбранные в нем слова: на каждой итерации обучения случайным образом выбирается абзац и в нем так же случайным образом выбирается заданное число слов, используемых затем для определения ошибки классификации и уточнения параметров (весов) НС. Обычно при обучении НС используется стохастический градиентный спуск, в котором градиенты получаются в результате обратного распространения ошибки [2].
Процесс создания doc2vec-модели на Python описывает следующая схема:

1. Загрузить корпус.
2. Создать список документов, снабженных индексами. Документы в этом списке представлены в виде списка слов.
3. Сформировать doc2vec-модель.
4. Сформировать по созданному в п. 2 списку словарь корпуса.
5. Обучить doc2vec-модель.
6. Сохранить модель в файл.

Пример реализации приведенной схемы:

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize as w_t
# 1
corp = [ # Корпус из 4-х документов (абзацев)
    'Заяц в лес бежал по лугу,',
    'Я из лесу шел домой, –',
    'Бедный заяц с перепугу',
    'Так и сел передо мной!']
e_corp = enumerate(corp) # class 'enumerate'
# Первый элемент e_corp (class 'tuple'):
# (0, 'Заяц в лес бежал по лугу,')
# 2
# tagged_corp - class 'list'
tagged_corp = [TaggedDocument(words = w_t(d.lower()), tags = [i]) for i, d in e_corp]
# Первый элемент tagged_corp (class 'gensim.models.doc2vec.TaggedDocument'):
# TaggedDocument(['заяц', 'в', 'лес', 'бежал', 'по', 'лугу', ','], [0])
# Список слов первого документа корпуса
doc_0_words = tagged_corp[0].words # class 'list'
# 3
# model - class 'gensim.models.doc2vec.Doc2Vec'
model = Doc2Vec(vector_size = 10, window = 3, min_count = 1, epochs = 100)
# 4
# Создаем словарь корпуса
model.build_vocab(tagged_corp)
# 5
# Обучение модели
model.train(tagged_corp, total_examples = model.corpus_count, epochs = model.epochs)
# 6
model.save('d2v.model')
```

Получение векторов слова и документа, заданного своим номером, а также предполагаемого вектора документа, заданного списком слов, иллюстрирует следующий *пример*:

```
from gensim.models.doc2vec import Doc2Vec
model = Doc2Vec.load('d2v.model') # Загрузка модели из файла
wv = model.wv # wv – индексированные векторы слов
print(wv['заяц']) # Координаты (с точностью 4 знака) слова 'заяц'
# [-0.0259 0.0247 -0.0158 0.0484 -0.0129 -0.0081 0.0233 -0.031 -0.0379 0.0284]
# Получаем словарь корпуса
vocab = wv.vocab # class 'dict'
# Первый элемент vocab: 'заяц'
# Вектор первого документа (с точностью 4 знака)
dv0 = model.docvecs.doctag_syn0[0]
# или:
#dv0 = model.docvecs[0]
# [-0.0113 0.0086 -0.032 0.0159 -0.0349 -0.021 -0.0205 -0.0018 0.0187 0.0221]
# Список слов первого документа корпуса
doc_0_words = ['заяц', 'в', 'лес', 'бежал', 'по', 'лугу', ',']
# Предполагаемый вектор документа, заданного списком doc_0_words
inferred_vector = model.infer_vector(doc_0_words) # class 'numpy.ndarray'
# inferred_vector (с точностью 4 знака):
# [-0.018 -0.0171 -0.0502 0.0472 -0.0472 0.0205 0.0296 0.0436 0.0426 -0.0146]
```

Замечание. Имена свойств класса, например, объекта model, можно получить следующим образом:

```
print(model.__dict__.keys())
```

Результат:

```
dict_keys(['sg', 'alpha', 'window', 'random', 'min_alpha', 'hs', 'negative', 'ns_exponent', 'cbow_mean', 'compute_loss',
'running_training_loss', 'min_alpha_yet_reached', 'corpus_count', 'corpus_total_words', 'vector_size', 'workers', 'epochs',
'train_count', 'total_train_time', 'batch_words', 'model_trimmed_post_training', 'callbacks', 'load', 'dbow_words', 'dm_concat',
'dm_tag_count', 'vocabulary', 'trainables', 'wv', 'docvecs', 'comment'])
```

5.10.2. Подготовка данных и результаты

Создание модели (реализация PV-DBOW):

```
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
```

```
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize as w_t
fn_xt, fn_xv, fn_wv = 'b_x_t.txt', 'b_x_v.txt', 'b_w2v.model'
sg, size, window, min_cnt, n_iter = 1, 150, 3, 1, 100
print('Создание doc2vec-модели по файлам', fn_xt, 'и', fn_xv)
# Работаем с фиксированным разбиением корпуса на обучающее и проверочное множества
x_trn = read_txt_f(fn_xt)
x_vl = read_txt_f(fn_xv)
x_trn.extend(x_vl) # Объединяем x_trn и x_vl и получаем полный корпус
tagged_corp = [TaggedDocument(words = w_t(d), tags = [str(i)])
               for i, d in enumerate(x_trn)]
model = Doc2Vec(vector_size = size,
                window = window, min_count = min_cnt, epochs = n_iter)
model.sg = sg # Реализацция PV-DBOW
# Создаем словарь корпуса
model.build_vocab(tagged_corp)
model.train(tagged_corp, total_examples = model.corpus_count,
            epochs = model.epochs)
model.save(fn_wv)
```

В отличие от word2vec, для представления документа берется вектор, поставляемый для него обученной doc2vec-моделью. Возможны несколько вариантов получения такого вектора:

```
from gensim.models.doc2vec import Doc2Vec
model = Doc2Vec.load(path + fn_wv)
wv = model.wv # wv – индексированные векторы слов
vocab = wv.vocab # Словарь модели
lst_corp, lst_cls = load_corp(fn_x, fn_y) # Загрузка документов корпуса и их меток
# Вариант 1 (основной):
lst_corp_codes = [dv for dv in model.docvecs.doctag_syn0]
# Вариант 2 (результат такой же, как и в варианте 1):
# lst_corp_codes = [] # Список векторных представлений документов
# n_doc = -1 # Номер документа
# for doc in lst_corp:
#     n_doc += 1
#     lst_corp_codes.append(model.docvecs[n_doc]) # Вектор документа
# Вариант 3 (основан на предполагаемых векторах):
#for doc in lst_corp:
#    doc = doc.split()
#    lst_corp_codes.append(model.infer_vector(doc)) # Предполагаемый вектор
```

Полный код загрузки модели, формирования векторов документов и обучения классификаторов:

```
import numpy as np
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
def classify(x_trn, y_trn, x_vl, y_vl):
    from sklearn.linear_model import SGDClassifier
    doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)
    doc_clf.fit(x_trn, y_trn) # Обучение классификатора
    print('Оценка точности классификации')
    score = doc_clf.score(x_vl, y_vl)
    print('Точность на проверочном множестве:', round(score, 4))
    score = doc_clf.score(x_trn, y_trn)
    print('Точность на обучающем множестве:', round(score, 4))
    #
    epochs = 90 # Число эпох обучения НС
    num_classes = 5 # Число классов
    def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
        from keras.models import Model
        from keras.layers import Input, Dropout, Dense
        import keras.utils as ut
        print('Преобразуем списки в массивы')
        x_trn = np.array(x_trn, dtype = 'float32')
        x_vl = np.array(x_vl, dtype = 'float32')
        # Переводим метки в one-hot представление
        y_trn = ut.to_categorical(y_trn, num_classes)
        y_vl = ut.to_categorical(y_vl, num_classes)
        inp_shape = (n_attrs_in_doc, )
        print('Формируем модель НС')
        inp = Input(shape = inp_shape, dtype = 'float32')
```

```
        x = Dropout(0.3)(inp)
        x = Dense(32, activation = 'relu')(x)
        output = Dense(num_classes, activation = 'softmax')(x)
        model = Model(inp, output)
        model.summary()
        model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        # Обучаем НС
        model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
                validation_data = (x_vl, y_vl))
        # Оценка модели НС на оценочных данных
        score = model.evaluate(x_vl, y_vl, verbose = 0)
        # Вывод потерь и точности
        print('Потери при тестировании: ', score[0])
        print('Точность при тестировании:', score[1])
    # После преобразования: x_trn, x_vl: class 'numpy.ndarray'
    n_attrs_in_doc = len(x_vl[0])
    print('Число признаков в документе:', n_attrs_in_doc)
    nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
fn_yt, fn_yv, fn_wv = 'b_y_t.txt', 'b_y_v.txt', 'b_w2v.model'
y_trn = read_txt_f(fn_yt, to_int = True)
y_vl = read_txt_f(fn_yv, to_int = True)
from gensim.models.doc2vec import Doc2Vec
model = Doc2Vec.load(fn_wv)
x_trn = [dv for dv in model.docvecs.doctag_syn0]
len_trn = len(y_trn)
x_vl = x_trn[len_trn:]
x_trn = x_trn[:len_trn]
#
# Полученные векторы документов передаются классификаторам – SGD и НС
# Классификация выполняется трижды
# Результаты (точность классификации) усредняются
for k in range(3):
    print('Номер попытки:', k + 1)
    classify(x_trn, y_trn, x_vl, y_vl)
```

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
***size* = 150**				
1	95.09	99.77	97.10	94.26
2	95.09	99.32	96.21	94.32
3	95.98	99.77	96.88	95.33
***size* = 768**				
1	95.76	100.0	98.21	99.72
2	96.65	100.0	97.99	99.66
3	95.98	100.0	97.77	99.61

5.11. Fasttext

5.11.1. Описание модели

В модели слово *w* ограничивается в модели символами "<" и >": *wf* = "<"+*w*+">".
Слово *w* представляется в fasttext виде множества, включающего *wf* и *n*-граммы из частей *wf*.

Пример множества, представляющего слово *бежит* с использованием триграмм:

<бежит>, <бе, беж, ежи, жит, ит>.

Введение ограничивающих символов позволяет различать *n*-граммы, например, *беж*, от одноименного слова, поскольку оно будет представлено в модели как *<беж>*.
На практике [13] одновременно используются *n*-граммы с $3 \le n \le 6$.
С такими значениями *n* получаем следующее представление слова *бежит*:
<бежит>, <бе, <беж, <бежи, <бежит, беж, бежи, бежит, бежит>, ежи, ежит, ежит>, жит, жит>, ит>.
В fasttext выполняется поиск векторных представлений всех полученных *n*-грамм, а вектор слова определяется как сумма векторов *n*-грамм этого слова.
Такой подход позволяет получить более достоверные векторы многих редко повторяемых слов – таких, *n*-граммы которых встречаются в корпусе чаще самих слов.
Для уменьшения издержек памяти *n*-граммы хешируются и слово представляется своим индексом в словаре корпуса и хеш-кодами своих *n*-грамм.
Векторы *n*-грамм получаются так же, как и в word2vec и doc2vec, в результате обучения НС предсказывать слово по контексту или контекст по слову.

5.11.2. Подготовка данных и результаты

При работе с fasttext библиотеки gensim модель создается и сохраняется в файл следующим образом:

```
from gensim.models import FastText
import multiprocessing
fn_c, fn_wv = 'b_x.txt', 'b_ft.model'
print('Создание fasttext-модели по файлу', fn_c)
sg, size, window, min_cnt, n_iter = 0, 150, 3, 1, 100
```

```
workers = multiprocessing.cpu_count()
model = FastText(corpus_file = fn_c, size = size, window = window,
                 min_count = min_cnt, sg = sg, workers = workers, iter = n_iter)
model.save(fn_wv)
```

После загрузки корпуса подготовку данных для классификаторов и их обучение выполнит следующий код:

```
import numpy as np
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
def read_corp():
    path = ''
    fn_xt, fn_yt, fn_d = 'b_x_t.txt', 'b_y_t.txt', 'b_dict.txt'
    fn_xv, fn_yv = 'b_x_v.txt', 'b_y_v.txt'
    x_trn = read_txt_f(path + fn_xt)
    y_trn = read_txt_f(path + fn_yt, to_int = True)
    x_vl = read_txt_f(path + fn_xv)
    y_vl = read_txt_f(path + fn_yv, to_int = True)
    # Список из слов словаря корпуса
    lst_dict = read_txt_f(fn_d)
    return x_trn, y_trn, x_vl, y_vl, lst_dict
def vocab_from_model(fn_wv):
    from gensim.models import FastText
    model = FastText.load(fn_wv) # FastText(vocab=51150, size=75, alpha=0.025)
    wv = model.wv # wv – индексированные векторы слов
    vocab = wv.vocab
    for w in vocab.keys():
        vec = wv[w]
        return wv, vocab, len(vec) # Модель, словарь модели и длина вектора слова
def make_trn_vl_4(x_trn_vl, wv, vocab, size):
    x_trn_vl_codes = [] # Список векторных представлений документов
    for doc in x_trn_vl:
        doc = doc.split()
        cw_sum = np.zeros(size)
        nv = nw = 0 # nw - число векторов, добавленных к cw_sum или в lst_codes
        for w in doc:
            in_vocab = vocab.get(w)
            if in_vocab is not None:
                cw = wv[w]
                cw_sum += cw
                nw += 1
        x_trn_vl_codes.append(cw_sum / nw)
    return x_trn_vl_codes
def classify(x_trn, y_trn, x_vl, y_vl):
    from sklearn.linear_model import SGDClassifier
    doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)
    doc_clf.fit(x_trn, y_trn) # Обучение классификатора
    print('Оценка точности классификации')
    score = doc_clf.score(x_vl, y_vl)
    print('Точность на проверочном множестве:', round(score, 4))
    score = doc_clf.score(x_trn, y_trn)
    print('Точность на обучающем множестве:', round(score, 4))
    #
    epochs = 90 # Число эпох обучения НС
    num_classes = 5 # Число классов
    def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
        from keras.models import Model
        from keras.layers import Input, Dropout, Dense
        import keras.utils as ut
        print('Преобразуем списки в массивы')
        x_trn = np.array(x_trn, dtype = 'float32')
        x_vl = np.array(x_vl, dtype = 'float32')
        # Переводим метки в one-hot представление
        y_trn = ut.to_categorical(y_trn, num_classes)
        y_vl = ut.to_categorical(y_vl, num_classes)
        inp_shape = (n_attrs_in_doc, )
        print('Формируем модель НС')
        inp = Input(shape = inp_shape, dtype = 'float32')
        x = Dropout(0.3)(inp)
        x = Dense(32, activation = 'relu')(x)
        output = Dense(num_classes, activation = 'softmax')(x)
        model = Model(inp, output)
        model.summary()
        model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

```
        # Обучаем НС
        model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
                validation_data = (x_vl, y_vl))
        # Оценка модели НС на оценочных данных
        score = model.evaluate(x_vl, y_vl, verbose = 0)
        # Вывод потерь и точности
        print('Потери при тестировании: ', score[0])
        print('Точность при тестировании:', score[1])
    # После преобразования: x_trn, x_vl: class 'numpy.ndarray'
    n_attrs_in_doc = len(x_vl[0])
    print('Число признаков в документе:', n_attrs_in_doc)
    nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
#
x_trn, y_trn, x_vl, y_vl, _ = read_corp()
wv, vocab, size = vocab_from_model('b_ft.model')
x_trn = make_trn_vl_4(x_trn, wv, vocab, size)
x_vl = make_trn_vl_4(x_vl, wv, vocab, size)
#
# Полученные векторы документов передаются классификаторам – SGD и НС
# Классификация выполняется трижды
# Результаты (точность классификации) усредняются
for k in range(3):
    print('Номер попытки:', k + 1)
    classify(x_trn, y_trn, x_vl, y_vl)
```

При использовании библиотеки fastext и обучения с учителем (train_supervised) предварительно обработанный корпус, в котором каждый документ предваряет строка __label__*n*, в которой *n* – номер класса документа, делится на обучающее и проверочное множества:

```
import random
lst_corp = read_txt_f(fn_x_lab)
random.shuffle(lst_corp) # Перемешиваем данные
len_lst = len(lst_corp)
len_trn = int(0.8 * len_lst)
len_tst = len_lst - len_trn
lst_trn = lst_corp[:len_trn]
lst_tst = lst_corp[len_trn:]
add_to_txt_f(lst_trn, fn_trn) # Запись данных в файлы
add_to_txt_f(lst_tst, fn_tst)
```

После этого создается и сохраняется модель:

```
import fasttext
hyper_params = {'lr': 0.01, 'epoch': 100, 'wordNgrams': 1, 'dim': size}
# Управляемое обучение
model = fasttext.train_supervised(input = fn_trn, **hyper_params)
model.save_model(fn_wv)
```

Перед формированием обучающих и проверочных данных модель загружается из файла и формируется ее словарь:

```
import fasttext
model = fasttext.load_model(fn_wv) # class 'fasttext.FastText._FastText'
all_words = model.get_words()
vocab = {}
wv = {}
for w in all_words:
    vocab[w] = 1
    wv[w] = model.get_word_vector(w)
```

Код дальнейшей подготовки данных полностью совпадает с выше приведенным кодом для fasttext библиотеки gensim.
Заметим, что в случае обучения без учителя (train_unsupervised), метки, предваряющие текст документа, должны быть устранены.
Как и в word2vec, вектор документа – это усредненные векторы слов.

Результаты (fasttext библиотеки gensim):

Номер попытки	SGD		НС	
	val_acc	*acc*	*val_acc*	*acc*
***size* = 150**				
1	96.88	98.65	97.10	95.05
2	97.32	98.71	97.10	96.17
3	97.10	98.20	97.10	95.95
***size* = 256**				
1	97.32	99.32	97.32	96.85
2	97.77	99.21	97.10	96.57
3	97.32	99.21	96.65	96.29
***size* = 300**				
1	97.54	99.27	97.32	96.85
2	97.32	98.99	97.99	96.40
3	96.43	98.26	97.54	97.07

5.11.3. Классификатор fasttext

Библиотека fasttext позволяет в случае обучения с учителем обратиться к файлам с обучающими и проверочными данными и оценить точность классификации на этих множествах:

```
acc = model.test(fn_trn)
tst_acc = model.test(fn_tst)
print('Точность на обучающем и проверочных множествах:',
      100 * round(acc[1], 4), 'и', 100 * round(tst_acc[1], 4))
```

Чтобы выполнить прогноз по каждому классу нужно предварительно убрать метку, предшествующую документу, а затем использовать метод predict:

```
# Освобождаемся от меток перед обращением к predict
x_tst_lab = read_txt_f(fn_tst)
x_tst = []
lst_n_cls = [0 for k in range(num_classes)]
lst_n_cls_true = lst_n_cls.copy()
for x in x_tst_lab:
    p = x.find(' ')
    lab = x[:p]
    cls = lab[-2:]
    if cls[0] == '_': cls = cls[1]
    cls = int(cls)
    x = x[p + 1:]
    p = model.predict(x, k = 1)
    lab_p = p[0][0]
    lst_n_cls[cls] += 1
    if lab == lab_p:
        lst_n_cls_true[cls] += 1
n = sum(lst_n_cls)
print('Точность прогноза на обучающем множестве размера', n)
print('По классам (%):')
for n_cls, cls in zip(range(num_classes), dict_cls.keys()):
    in_cls = lst_n_cls[n_cls]
    acc_cls = 100 * round(lst_n_cls_true[n_cls] / in_cls, 4)
    print('{} - {}({}): {}'.format(n_cls, cls, in_cls, acc_cls))
print('Усредненная точность (%):', 100 * round(sum(lst_n_cls_true) / n, 4))
```

Возможный результат:

```
Точность прогноза (%) по классам на проверочном множестве размера 647:
0 - автомобили (43): 0.0
1 - здоровье (22): 0.0
2 - культура (79): 0.0
3 - наука (48): 0.0
4 - недвижимость (17): 0.0
5 - политика (92): 100.0
6 - происшествия (83): 51.81
7 - реклама (12): 8.33
8 - семья (24): 8.33
9 - спорт (65): 0.0
10 - страна (51): 0.0
11 - техника (64): 0.0
12 - экономика (47): 0.0
Усредненная точность (%): 21.33
```

5.12. GloVe

5.12.1. Описание модели

GloVe (глобальные векторы) – это алгоритм обучения без учителя для получения векторных представлений слов [14]. Обучение основано на статистике встречаемости слов корпуса.
В модели в результате анализа корпуса вычисляются вероятности p_{ik} появления слова i в контексте слова k. Общее представление модели GloVe описывается следующим соотношением:

$$F\left(w_i, w_j, \overline{w}_k\right) = p_{ik}/p_{jk},$$

в котором $w_i, w_j, \overline{w}_k$ – соответственно векторы слов i, j и контекстные векторы.
Вероятности извлекаются из корпуса:

$p_{ik} = X_{ik} / X_i$,

где X_{ik} – число появлений слова i в контексте слова k, а X_i – число появлений слова i в корпусе.
Функция F интерпретируются в [14] как экспонента с аргументом

$$x = (w_i - w_j)^T \overline{w}_k = w_i^T \overline{w}_k - w_j^T \overline{w}_k,$$

$$e^x = e^{w_i^T \overline{w}_k} / e^{w_j^T \overline{w}_k} = p_{ik}/p_{jk}.$$

Что позволяет, в частности, записать следующее выражение:

$$w_i^T \bar{w}_k + b_i + \bar{b}_k = \log X_{ik},$$

на основании которого записывается линейная модель:

$$w_i^T \bar{w}_k + b_i + \bar{b}_k = \log X_{ik},$$

где свободный член $b_i = \log X_i$, а $\bar{b}_k$ добавлен для симметрии.
Исходя из этой модели, в [14] формулируется целевая функция – взвешенная сумма квадратов отклонений, значение которой при обучении минимизируется:

$$J = \sum_{i,j}^{V} f(X_{ij})(w_i^T \bar{w}_k + b_i + \bar{b}_k - \log X_{ik})^2,$$

где V – размер словаря;
$f(x)$ – весовая функция, в качестве которой в [14] предлагается следующая:

$$f(x) = \begin{cases} (x/x_{max})^{\alpha}, & x < x_{max} \\ 1, & x \geq x_{max} \end{cases}.$$

В качестве x_{max} можно взять, например, число 100, а α – 3/4 [14].
Экспериментально установлено, что малое расстояние между векторами GloVe указывают на семантическую близость соответствующих слов.

5.12.2. Подготовка данных и результаты

При работе с англоязычными текстами можно воспользоваться готовыми векторами GloVe, полученными на различных корпусах [15].
Векторы хранятся в текстовых файлах. Каждая строка файла начинается с имени токена, вслед за которым следуют координаты вектора, представляющего токен, например:

the 0.418 0.24968 -0.41242 0.1217 0.34527 -0.044457 -0.49688 ... -0.78581
, 0.013441 0.23682 -0.16899 0.40951 0.63812 0.47709 -0.42852 ... 0.30392

По загруженному из [15] файлу, как и в случае предобученной word2vec модели, формируется файл, содержащий только векторы тех слов, которые имеются в словаре BBCD (см. word2vec).
Отличие только в именах файлов. Вместо

```
f = open('word2vec/model.txt', 'r', encoding = 'utf-8')
f2 = open('word2vec/b_w2v_txt.txt', 'w', encoding = 'utf-8')
```

указываем

```
f = open('glove/glove_6B_300d.txt', 'r', encoding = 'utf-8')
f2 = open('glove/b_glove.txt', 'w', encoding = 'utf-8')
```

Код, формирующий словарь *wv* с элементами {*токен* : *вектор*}:

```
wv = {}
with open(fn_wv, 'r', encoding = 'utf-8') as f:
    for vec in f:
        vec = vec.split()
        word = vec[0] # Токен из файла fn_wv
        if dict_x.get(word) is not None:
            vec = np.array(vec[1:], dtype = 'float32')
            wv[word] = vec
```

В этом коде dict_x – это словарь корпуса. Присутствие dict_x ограничивает *wv* только теми элементами, токены которых присутствуют в используемом корпусе.
Далее, как и в других моделях, по словарю *wv* создаются обучающее и проверочное множества, в которых токены представлены векторами GloVe.
Полный код подготовки и использования данных:

```
import numpy as np
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
def read_corp():
    path = ''
    fn_xt, fn_yt, fn_d = 'b_x_t.txt', 'b_y_t.txt', 'b_dict.txt'
    fn_xv, fn_yv = 'b_x_v.txt', 'b_y_v.txt'
    x_trn = read_txt_f(path + fn_xt)
    y_trn = read_txt_f(path + fn_yt, to_int = True)
    x_vl = read_txt_f(path + fn_xv)
    y_vl = read_txt_f(path + fn_yv, to_int = True)
```

```
    # Список из слов словаря корпуса
    lst_dict = read_txt_f(fn_d)
    return x_trn, y_trn, x_vl, y_vl, lst_dict
def read_glove_wv(fn_x):
    corp = read_txt_f(fn_x)
##    size = 200
##    fn_wv = 'glove/glove_6B_' + str(size) + 'd.txt'
    size = 300     fn_wv = 'glove/b_glove.txt'     dict_x = {}
    for d in corp:
        d = d.split()
        for w in d:
            dict_x[w] = 1
    wv = {}
    with open(fn_wv, 'r', encoding = 'utf-8') as f:
        for vec in f:
            vec = vec.split()
            word = vec[0] # Токен из файла fn_wv
            if dict_x.get(word) is not None:
                vec = np.array(vec[1:], dtype = 'float32')
                wv[word] = vec
    print('Размер словаря:', len(wv))
    return wv, size
def make_trn_vl_glove(x_trn_vl, wv, size):
    x_trn_vl_codes = [] # Список векторных представлений документов
    for doc in x_trn_vl:
        doc = doc.split()
        cw_sum = np.zeros(size)
        nw = 0 # nw - число векторов, суммированных в cw_sum
        for w in doc:
            in_vocab = wv.get(w)
            if in_vocab is not None:
                cw_sum += in_vocab
                nw += 1
        x_trn_vl_codes.append(cw_sum / nw)
    return x_trn_vl_codes
def classify(x_trn, y_trn, x_vl, y_vl):
    from sklearn.linear_model import SGDClassifier
    doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)
    doc_clf.fit(x_trn, y_trn) # Обучение классификатора
    print('Оценка точности классификации')
    score = doc_clf.score(x_vl, y_vl)
    print('Точность на проверочном множестве:', round(score, 4))
    score = doc_clf.score(x_trn, y_trn)
    print('Точность на обучающем множестве:', round(score, 4))
    #
    epochs = 90 # Число эпох обучения НС
    num_classes = 5 # Число классов
    def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
        from keras.models import Model
        from keras.layers import Input, Dropout, Dense
        import keras.utils as ut
        print('Преобразуем списки в массивы')
        x_trn = np.array(x_trn, dtype = 'float32')
        x_vl = np.array(x_vl, dtype = 'float32')
        # Переводим метки в one-hot представление
        y_trn = ut.to_categorical(y_trn, num_classes)
        y_vl = ut.to_categorical(y_vl, num_classes)
        inp_shape = (n_attrs_in_doc, )
        print('Формируем модель НС')
        inp = Input(shape = inp_shape, dtype = 'float32')
        x = Dropout(0.3)(inp)
        x = Dense(32, activation = 'relu')(x)
        output = Dense(num_classes, activation = 'softmax')(x)
        model = Model(inp, output)
        model.summary()
        model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        # Обучаем НС
        model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
                  validation_data = (x_vl, y_vl))
        # Оценка модели НС на оценочных данных
        score = model.evaluate(x_vl, y_vl, verbose = 0)
        # Вывод потерь и точности
        print('Потери при тестировании: ', score[0])
        print('Точность при тестировании:', score[1])
    # После преобразования: x_trn, x_vl: class 'numpy.ndarray'
    n_attrs_in_doc = len(x_vl[0])
    print('Число признаков в документе:', n_attrs_in_doc)
    nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
#
fn_x = 'b_x.txt'
x_trn, y_trn, x_vl, y_vl, _ = read_corp()
wv, size = read_glove_wv(fn_x)
```

```
x_trn = make_trn_vl_glove(x_trn, wv, size)
x_vl = make_trn_vl_glove(x_vl, wv, size)
#
# Полученные векторы документов передаются классификаторам – SGD и НС
# Классификация выполняется трижды
# Результаты (точность классификации) усредняются
for k in range(3):
    print('Номер попытки:', k + 1)
    classify(x_trn, y_trn, x_vl, y_vl)
```

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
size = 200				
1	97.77	98.09	97.77	96.40
2	96.88	97.86	97.77	96.90
3	97.54	97.58	97.77	96.74
size = 300				
1	96.65	97.92	97.77	97.86
2	98.66	98.37	97.99	97.81
3	96.65	97.86	97.99	97.86

5.13. Введение в модели на архитектуре Трансфомер

5.13.1. Порядок использования моделей

Модели с архитектурой Трансфомер – это предобученные НС, позволяющие решать широкий круг задач обработки естественного языка (ЕЯ).
Большинство моделей реализуют преобразование seq2seq (*последовательность в последовательность*), немногие – tetx2text (текст в текст).
В общем случае доступны несколько предобученных версий моделей, например, при работе с BERT можно употребить bert-base-uncased, bert-large-uncased, bert-base-cased, bert-large-cased, bert-base-chinese, bert-base-german-cased и другие [16], различающихся числом слоев и параметров и обученных на разных корпусах.
В работе берутся базовые версии моделей. Размер выхода в таких версиях равен 768.
В рассматриваемой примере классификации документов модели используются для получения векторов документов, которые формируются в результате усреднения векторов токенов, выделенных в документе.
Далее полученные векторы загружаются программой, классифицирующей документы.

5.13.2. Программа получения векторов документов

Программа применяется при работе со всеми моделями на архитектуре Трансформер.
Выбор модели регулирует параметр *m_num*; выбор вида множества (обучающее или проверочное) – *tv*.

```
!pip install transformers
from sys import exit
import torch, time
from google.colab import drive
drv = '/content/drive/'
drive.mount(drv)
path = drv + 'My Drive/bert/'
nw_max = 512
vec_len = 768 # 1280, если, например, gpt2-large; 512 - MobileBert
# 0 - GPT2 (g); 1 - Bert (b); 2 - Albert (al); 3 - Bart (ba)
# 4 - BertGeneration (bg); 5 - ConvBert (cb); 6 - T5 (t5)
# 7 - Roberta (ro); 8 - Deberta (de); 9 - DistilBert (di)
# 10 - Electra (e); 11 - Funnel (f); 12 - LED (l); 13 - Longformer (lo)
# 14 - MobileBert (mb); 15 - MPNet (mp); 16 - mT5 (mt); 17 - Reformer (re)
# 18 - SqueezeBert (sb); 19 - Tapas (tp); 20 - XLMRoberta (xr); 21 - XLNet (xn)
m_num = 20
tv = 1 # 0 - train; 1 - test; 2 - весь набор
show_model = not True
if m_num in [2, 6, 20, 21]: # Albert, T5, XLMRoberta, XLNet
  !pip install sentencepiece
  # vocab_file = path + 'b_sp.model'
if m_num in [3, 6, 12]: # Bart; T5; LED
  from_encoder = True
if m_num == 0: # Размер словаря: 50257
  from transformers import GPT2Tokenizer as tkn
  from transformers import GPT2ForSequenceClassification as mdl
  # from transformers import GPT2LMHeadModel as mdl
  pretrained_mdl = 'gpt2' # gpt2, gpt2-large - версии обученной модели
elif m_num == 1:
  from transformers import BertTokenizer as tkn
  from transformers import BertForSequenceClassification as mdl
  # from transformers import BertModel as mdl
  pretrained_mdl = 'bert-base-uncased'
elif m_num == 2: # Размер словаря: 30000
  from transformers import AlbertTokenizer as tkn
  from transformers import AlbertForSequenceClassification as mdl
  # from transformers import AlbertModel as mdl
```

```
  pretrained_mdl = 'albert-base-v2'
elif m_num == 3: # Размер словаря: 50265
  from transformers import BartTokenizer as tkn
  from transformers import BartForSequenceClassification as mdl
  # from transformers import BartModel as mdl
  pretrained_mdl = 'facebook/bart-base' # bart-large
elif m_num == 4:
  from transformers import BertGenerationEncoder as mdl, BertTokenizer as tkn
  pretrained_mdl = 'bert-base-uncased'
elif m_num == 5: # Размер словаря: 29514
  from transformers import ConvBertTokenizer as tkn
  from transformers import ConvBertForSequenceClassification as mdl
  # from transformers import ConvBertModel as mdl
  pretrained_mdl = 'YituTech/conv-bert-base'
elif m_num == 6: # Размер словаря: 32128
  from transformers import T5Tokenizer as tkn
  if from_encoder:
    from transformers import T5EncoderModel as mdl
  else:
    from transformers import T5Model as mdl
  # t5-small:6; t5-base:12; t5-large:24; t5-3b:24; t5-11b:24
  # 6, 12, 24 - число модулей внимания
  pretrained_mdl = 't5-base'
elif m_num == 7: # Размер словаря: 50265
  from transformers import RobertaTokenizer as tkn
  from transformers import RobertaForSequenceClassification as mdl
  pretrained_mdl = 'roberta-base'
elif m_num == 8: # Размер словаря: 50265
  from transformers import DebertaTokenizer as tkn
  from transformers import DebertaForSequenceClassification as mdl
  # from transformers import DebertaModel as mdl
  pretrained_mdl = 'microsoft/deberta-base'
elif m_num == 9: # Размер словаря: 30522
  from transformers import DistilBertTokenizer as tkn
  from transformers import DistilBertForSequenceClassification as mdl
  # from transformers import DistilBertModel as mdl
  pretrained_mdl = 'distilbert-base-uncased'
elif m_num == 10: # Размер словаря: 30522
  from transformers import ElectraTokenizer as tkn
  from transformers import ElectraForSequenceClassification as mdl
  pretrained_mdl = 'google/electra-base-discriminator'
elif m_num == 11: # Размер словаря: 30522
  from transformers import FunnelTokenizer as tkn
  from transformers import FunnelForSequenceClassification as mdl
  pretrained_mdl = 'funnel-transformer/medium-base'
elif m_num == 12: # Размер словаря: 50265
  from transformers import LEDTokenizer as tkn
  from transformers import LEDForSequenceClassification as mdl
  pretrained_mdl = 'allenai/led-base-16384'
  nw_max = 1024
elif m_num == 13: # Размер словаря: 50265
  from transformers import LongformerTokenizer as tkn
  from transformers import LongformerForSequenceClassification as mdl
  pretrained_mdl = 'allenai/longformer-base-4096'
  nw_max = 1024
elif m_num == 14: # Размер словаря: 30522
  from transformers import MobileBertTokenizer as tkn
  from transformers import MobileBertForSequenceClassification as mdl
  pretrained_mdl = 'google/mobilebert-uncased'
  vec_len = 512
elif m_num == 15: # Размер словаря: 30527
  from transformers import MPNetTokenizer as tkn
  from transformers import MPNetForSequenceClassification as mdl
  pretrained_mdl = 'microsoft/mpnet-base'
elif m_num == 16: # Размер словаря:
  from transformers import T5Tokenizer as tkn
  from transformers import MT5EncoderModel as mdl
  pretrained_mdl = 'google/mt5-base'
elif m_num == 17: # Размер словаря:
  from transformers import ReformerTokenizer as tkn
  from transformers import ReformerForSequenceClassification as mdl
  pretrained_mdl = 'google/reformer-crime-and-punishment' # google/reformer-enwik8
elif m_num == 18: # Размер словаря: 30528
  from transformers import SqueezeBertTokenizer as tkn
  from transformers import SqueezeBertForSequenceClassification as mdl
  pretrained_mdl = 'squeezebert/squeezebert-uncased'
elif m_num == 19: # Размер словаря: 30522
  !pip install torch-scatter -f https://pytorch-geometric.com/whl/torch-1.7.0+cu101.html
  import pandas as pd, torch_scatter
  from transformers import TapasTokenizer as tkn
  from transformers import TapasForSequenceClassification as mdl
  pretrained_mdl = 'google/tapas-base'
```

```
elif m_num == -20: # Размер словаря: 267'735
  from transformers import TransfoXLTokenizer as tkn
  from transformers import TransfoXLForSequenceClassification as mdl
  pretrained_mdl = 'transfo-xl-wt103'
  vec_len = 1024
elif m_num == 20: # Размер словаря: 250'002
  from transformers import XLMRobertaTokenizer as tkn
  from transformers import XLMRobertaForSequenceClassification as mdl
  pretrained_mdl = 'xlm-roberta-base'
elif m_num == 21: # Размер словаря: 32000
  from transformers import XLNetTokenizer as tkn
  from transformers import XLNetForSequenceClassification as mdl
  pretrained_mdl = 'xlnet-base-cased'
pretrained_tkn = pretrained_mdl
#
data_set = 2 # 0 - ru; 1 - reuters; 2 - bbc; 3 - docs; 4 - e_docs; 5 - sentiments
#
lst_pre_ds = ['', 'r_', 'b_', 'd_', 'e_', 's_']
pre_ds = lst_pre_ds[data_set]
if tv == 0:
    suf = '_t'
elif tv == 1:
    suf = '_v'
else:
    suf = ''
fn_b = path + pre_ds + 'x' + suf + '.txt'
fn_b_vecs = ['g', 'b', 'al', 'ba', 'bg', 'cb', 't5', 'ro', 'de', 'di', 'e', 'f',
             'le', 'lo', 'mb', 'mp', 'mt', 're', 'sb', 'tp', 'xr', 'xn']
fn_b_vecs = fn_b_vecs[m_num]
if m_num in [3, 6, 12]: # Bart; T5; LED
  fn_b_vecs += ('e' if from_encoder else 'd')
fn_b_vecs = pre_ds + fn_b_vecs + '_vecs' + suf + '.txt'
#
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
def find_str_vec(vec):
  str_vec = ''
  for v in vec:
    v = float(v.numpy())
    str_vec += (str(round(v, 4)) + ' ')
  return str_vec.rstrip()
#
print('m_num =', m_num, '/ tv =', tv, '/ vec_len =', vec_len)
print('Модель:', pretrained_mdl, '/', pretrained_tkn)
# Создаем токенизатор
if m_num == -2:
  tokenizer = tkn(vocab_file)
else:
  tokenizer = tkn.from_pretrained(pretrained_tkn)
  vocab = tokenizer.get_vocab()
  print('Размер словаря:', len(vocab))
t0 = time.time()
lst_txt = read_txt_f(fn_b)
lst_tokenized = []
if m_num in [-1, -2]: # Прежний вариант
  vocab = tokenizer.get_vocab()
  for d in lst_txt:
    d2 = d.split()
    d = ''
    nw = 0
    for w in d2:
      if vocab.get(w) is not None:
        nw += 1
        d += w + ' '
        if nw == nw_max: break
    d = d.rstrip()
    dt = tokenizer(d, return_tensors = "pt")
    lst_tokenized.append(dt)
elif m_num == -3:
  for d in lst_txt:
    dt = tokenizer(d, return_tensors = "pt")
    lst_tokenized.append(dt)
elif m_num == 19: # Tapas
  for d in lst_txt:
    data = {'sen': [d]}
    table = pd.DataFrame.from_dict(data)
```

```
    queries = []
    dt = tokenizer(table = table, queries = 'sen')['input_ids']
    if len(dt) > nw_max: dt = dt[:nw_max]
    dt = torch.tensor([dt])
    lst_tokenized.append(dt)
else: # Текущий вариант (все остальные модели)
  for d in lst_txt:
    dt = tokenizer(d)['input_ids']
    if len(dt) > nw_max: dt = dt[:nw_max]
    if m_num in [3, 12]:
      n_eos = dt.count(2) # eos
      if n_eos == 0:
        dt[nw_max - 1] = 2
    dt = torch.tensor([dt])
    lst_tokenized.append(dt)
t2 = time.time()
print('Время получения индексов:', round(t2 - t0, 2))
model = mdl.from_pretrained(pretrained_mdl, output_hidden_states = True)
if show_model: print(model)
model.eval()
print('Формирование файла', fn_b_vecs)
fb = open(path + fn_b_vecs, 'w', encoding = 'utf-8') # Результирующий файл
n_vecs = 0
batch_i = 0
for dt in lst_tokenized:
  with torch.no_grad():
    if m_num in [-1, -2, -3]: # Прежний вариант
      outputs = model(**dt)
    elif m_num not in [6, 12]:
      outputs = model(dt)
    else: # T5, LED
      if from_encoder:
        outputs = model(input_ids = dt)
      else:
        outputs = model(input_ids = dt, decoder_input_ids = dt)
  if m_num in [0, 1, 2, 5, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21]:
    hidden_states = outputs.hidden_states # len(hidden_states) = 13
    last_hidden_state = hidden_states[len(hidden_states) - 1]
  elif m_num in [3, 12]: # Bart; LED
    if from_encoder:
      last_hidden_state = outputs.encoder_last_hidden_state
    else:
      hidden_states = outputs.decoder_hidden_states
      last_hidden_state = hidden_states[len(hidden_states) - 1]
      # last_hidden_state = outputs.last_hidden_state # *Model
  else:
    last_hidden_state = outputs.last_hidden_state
  sz = last_hidden_state.size()
  sz_1 = sz[1]
  vec_sum = torch.zeros(vec_len)
  for token_i in range(sz_1):
      vec = last_hidden_state[batch_i][token_i]
      vec_sum += vec
  vec_sum /= sz_1
  str_vec = find_str_vec(vec_sum)
  fb.write(str_vec + '\n')
  n_vecs += 1
  if n_vecs % 200 == 0:
    print('Число векторов в файле:', n_vecs)
fb.close()
print('Всего векторов в файле:', n_vecs)
print('Время получения выходов блоков модели:', round(time.time() - t2, 2))
#
# tokenizer = tkn.from_pretrained(pretrained_mdl)
# vocab = tokenizer.get_vocab()
# print(len(vocab)) # 50265
# print(vocab.get('man'))
# model = mdl.from_pretrained(pretrained_mdl)
# inputs = tokenizer("Hello, my dog is cute", return_tensors="pt")
# outputs = model(**inputs)
# last_hidden_states = outputs.last_hidden_state
# print(last_hidden_states.shape) # torch.Size([1, 8, 768])
# exit()
#
#tokenizer = GPT2Tokenizer.from_pretrained(('gpt2')
#t_inds = tokenizer.encode('man and woman', add_prefix_space = False)
#model = GPT2LMHeadModel.from_pretrained((pretrained_mdl)
#word_embeddings = model.transformer.wte.weight
#d_vec = torch.zeros(vec_len)
#with torch.no_grad():
#  vec = word_embeddings[t_inds[0]]
#  d_vec += vec
```

```
#  vec = word_embeddings[t_inds[1]]
#  d_vec += vec
#  vec = word_embeddings[t_inds[1]]
#  d_vec += vec
#print(d_vec / 3)
#exit()
#
# word_embeddings = model.transformer.wte.weight  # Word Token Embeddings
# position_embeddings = model.transformer.wpe.weight  # Word Position Embeddings
# t_inds = tokenizer.encode('man and woman', add_prefix_space = True)
# print(t_inds)
# t_inds = tokenizer.encode('to man in a boat', add_prefix_space = True)
# print(t_inds)
# vec = word_embeddings[t_inds[0],:]
# print(len(vec))
# print(type(word_embeddings)) # class 'torch.nn.parameter.Parameter'
# with torch.no_grad():
#  for we in word_embeddings:
#    vec = we[0]
#    print(vec)
#    break
# id = tokenizer.encode(w, add_prefix_space = False)
# id = tokenizer(w)['input_ids']
```

5.13.3. Программа загрузки векторов и классификации документов

Полученные с использованием модели векторы документов загружаются и передаются классификатором.
Каждый классификатор запускается трижды. Выбор модели регулирует параметр *m_num*.

```
import numpy as np
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # class 'list'
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
def add_to_trn_vl(bert_vecs):
    x_trn_vl = []
    for vec in bert_vecs:
        vec = vec.split()
        x_trn_vl.append([float(v) for v in vec])
    return x_trn_vl
def classify(x_trn, y_trn, x_vl, y_vl):
    from sklearn.linear_model import SGDClassifier
    doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)
    doc_clf.fit(x_trn, y_trn) # Обучение классификатора
    print('Оценка точности классификации')
    score = doc_clf.score(x_vl, y_vl) # class 'numpy.float64'
    print('Точность на проверочном множестве:', round(score, 4))
    score = doc_clf.score(x_trn, y_trn) # class 'numpy.float64'
    print('Точность на обучающем множестве:', round(score, 4))
    #
    epochs = 90 # Число эпох обучения НС
    num_classes = 5 # Число классов
    def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
        from keras.models import Model
        from keras.layers import Input, Dropout, Dense
        import keras.utils as ut
        print('Преобразуем списки в массивы')
        x_trn = np.array(x_trn, dtype = 'float32')
        x_vl = np.array(x_vl, dtype = 'float32')
        # Переводим метки в one-hot представление
        y_trn = ut.to_categorical(y_trn, num_classes)
        y_vl = ut.to_categorical(y_vl, num_classes)
        inp_shape = (n_attrs_in_doc, )
        print('Формируем модель НС')
        inp = Input(shape = inp_shape, dtype = 'float32')
        x = Dropout(0.3)(inp)
        x = Dense(32, activation = 'relu')(x)
        output = Dense(num_classes, activation = 'softmax')(x)
        model = Model(inp, output)
        model.summary()
        model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        # Обучаем НС
        model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
                  validation_data = (x_vl, y_vl))
        # Оценка модели НС на оценочных данных
        score = model.evaluate(x_vl, y_vl, verbose = 0)
```

```
        # Вывод потерь и точности
        print('Потери при тестировании: ', score[0])
        print('Точность при тестировании:', score[1])
    # После преобразования: x_trn, x_vl: class 'numpy.ndarray'
    n_attrs_in_doc = len(x_vl[0])
    print('Число признаков в документе:', n_attrs_in_doc)
    nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
#
m_num = 21 # 3: 'bae' или 'bad'
fn_b_vecs = ['g', 'b', 'al', 'bad', 'bg', 'cb', 't5', 'ro', 'de', 'di', 'e', 'f',
             'led', 'lo', 'mb', 'mp', 'mt', 're', 'sb', 'tp', 'xr', 'xn']
model_nm = ['GPT-2', 'Bert', 'Albert', 'Bart', ' BertGeneration', 'ConvBert',
            'T5', 'Roberta', 'Deberta', 'Distilbert', 'Electra', 'Funnel',
            'LED', 'Longformer', 'MobileBert', 'MPNet', 'MT5', ' Reformer',
            'SqueezeBert', 'Tapas', 'XLMRoberta', 'XLNet']
fn_yt, fn_yv = 'b_y_t.txt', 'b_y_v.txt'
y_trn = read_txt_f(fn_yt, to_int = True)
y_vl = read_txt_f(fn_yv, to_int = True)
fn_b_v = 'b_' + fn_b_vecs[m_num] + '_vecs'
if m_num == -1: # Bert
    suffs = ['_cls', '', '4', 'sep']
    fn_b_vecs_t = [fn_b_v + sf + '_t.txt' for sf in suffs]
    fn_b_vecs_v = [fn_b_v + sf + '_v.txt' for sf in suffs]
    model_nm = ['Bert' + sf for sf in suffs]
elif m_num in [-3, -6, -12]: # Bart, T5, LED
    suffs = ['e', 'd']
    fn_b_vecs_t = [fn_b_v + sf + '_t.txt' for sf in suffs]
    fn_b_vecs_v = [fn_b_v + sf + '_v.txt' for sf in suffs]
    model_nm = ['Bart' + sf for sf in suffs]
else:
    fn_b_vecs_t = [fn_b_v + '_t.txt']
    fn_b_vecs_v = [fn_b_v + '_v.txt']
    model_nm = [model_nm[m_num]]
for fn_b_t, fn_b_v, nm in zip(fn_b_vecs_t, fn_b_vecs_v, model_nm):
    print(nm)
    bert_vecs_t = read_txt_f(fn_b_t)
    bert_vecs_v = read_txt_f(fn_b_v)
    x_trn = add_to_trn_vl(bert_vecs_t)
    x_vl = add_to_trn_vl(bert_vecs_v)
    #
    # Полученные векторы документов передаются классификаторам – SGD и НС
    # Классификация выполняется трижды
    # Результаты (точность классификации) усредняются
    for k in range(3):
        print('Номер попытки:', k + 1)
        classify(x_trn, y_trn, x_vl, y_vl)
```

5.13.4. Сравнительная оценка моделей

Модели программной обработки ЕЯ принято сравнивать на данных, предоставляемых надежными источниками, например, GLUE [17], SquAD – The Stanford Question Answering Dataset [18] и RACE [19] – набор данных для оценки методов понимания текста (reading comprehension).
GLUE предоставляет наборы данных для решения следующих задач (в скобках указана метрика):

- The Corpus of Linguistic Acceptability (Matthew's Corr) – проверка лингвистической целостности.
- The Stanford Sentiment Treebank (Accuracy) – анализ тональности.
- Microsoft Research Paraphrase Corpus (F1 / Accuracy) – анализ семантической эквивалентности пар фраз.
- Semantic Textual Similarity (STS) Benchmark (Pearson-Spearman Corr) – анализ семантического сходства текстов.
- Quora Question Pairs (F1 / Accuracy) – определение эквивалентности двух вопросов (*пример* разных вопросов: "What are natural numbers?" – "What is a least natural number?").
- MultiNLI Matched (Accuracy) – многозначные заключения на ЕЯ (истинные).
- MultiNLI Mismatched (Accuracy) – многозначные заключения на ЕЯ (ложные).
- Question NLI (Accuracy) – ответы на вопросы.
- Recognizing Textual Entailment (Accuracy) – распознавание, является ли смысл одного текста вытекающим из другого текста.
- Winograd NLI (Accuracy) – разрешение анафоры (anaphora resolution), то есть выяснение, к чему относится местоимение в тексте.
  Пример:
  The city councilmen refused the demonstrators a permit because **they** [feared/advocated] violence.
  Если в предложении feared, то they относится к city council; если – advocated, то they отнсится к demonstrators.
- Diagnostics Main (Matthew's Corr) – выявления монотонности утверждений, анализ смысла слов.

5.13.5. Трансформер

Приводимые в разделе сведения и рисунки почерпнуты из [20].

Введение

Рекуррентные НС нередко применяют для обработки последовательностей. НС генерируют последовательность скрытых состояний h_t как функцию скрытых состояний h_{t-1} и входа в момент времени t. Последовательная природа описанного процесса затрудняет параллельную обработку обучающих примеров.
Механизмы внимания позволяют отслеживать зависимости в произвольно длинных последовательностях. В большом числе

случаев такие механизмы применяются в рекуррентных НС, что возвращает к вышеописанной проблеме производительности модели.
Архитектура трансформера более благоприятна для распараллеливания. Она исключает рекуррентную обработку данных, полностью полагаясь на механизм внимания, способствующий установлению имеющихся в данных зависимостей.

Общие положения

В трансформере, в отличие от других моделей, уменьшающих объем последовательных вычислений, например ConvS2S и ByteNet, число операций, связывающих произвольные позиции входа (выхода) фиксировано. (В ConvS2S оно растет линейно, а в ByteNet – логарифмически от расстояния между позициями.) При этом, однако, снижается уровень детализации из-за усреднения полученных в результате применения операций внимания весов позиций. Этот недостаток нивелируется использованием многоголового внимания.
Самовнимание, называемое иногда внутренним, – это механизм, связывающий различные позиции одной последовательности при вычислении ее представления.
Трансформер – это первая трансдукционная (преобразующая) модель, полностью полагающаяся на самовнимание при вычислении представлений входных данных без использования, в отличие от рекуррентных и сверточных моделей, последовательных операций.

Архитектура модели

Большинство эффективных преобразующих моделей относятся к классу кодер-декодер.
Кодер преобразует последовательность $(x_1, ..., x_n)$ в последовательность z = $(z_1, ..., z_n)$. Декодер, получив z, генерирует (по одному элементу) выходную последовательность $(y_1, ..., y_m)$.
На каждом шаге модель авторегрессионная: потребляет ранее сгенерированный элемент в качестве дополнительного входа при генерировании следующего элемента.
Трансформер следует этой схеме, совмещая и в кодере, и в декодере, самовнимание и полносвязанные слои (рис. 6).

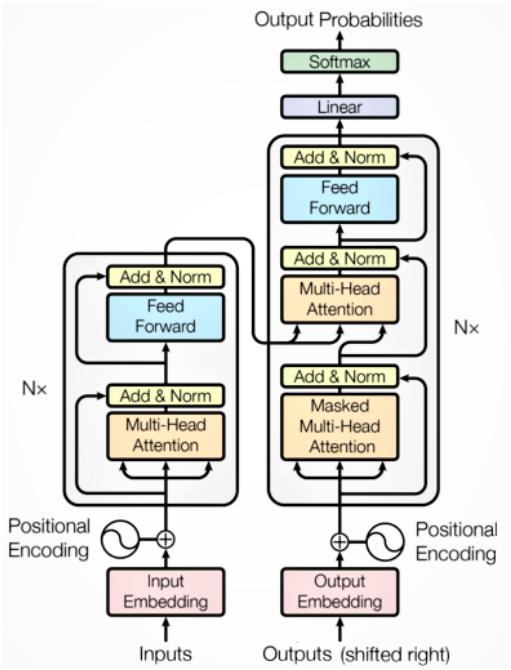

Рис. 6. Архитектура трансформера [20]

Кодер и декодер

Кодер содержит N (N = 6) одинаковых составных слоев (блоков). В каждом блоке два обучаемых подслоя. Первый реализует механизм многоголового внимания (самовнимания), второй – полносвязная нейронная сеть прямого распространения. Каждый из этих подслоев охватывает остаточное соединение, поступающее на слой нормализации LayerNorm, на выходе которого имеем LayerNorm(x + Sublayer(x)), где x – данные поступающие на подслой Sublayer, и они-же передаются по остаточной ветви.
Чтобы упростить остаточные соединения, каждый подслой, как и эмбединг-слои, имеет размер выхода, равный 512.
В **декодере** так же N одинаковых блоков, но уже с тремя подслоями каждый.
Два (на рис. 6 – это второй и третий по счету) таких же, как и в кодере, а первый применяет многоголовое внимание к выходу кодера. Как и в кодере, каждый подслой снабжен остаточным соединением, приходящим на слой нормализации.
Слои самовнимания устроены таким образом, что в текущей позиции запрещается посещать последующие позиции. Это в сочетании с тем, что выходные эмбединги смещены на одну позицию, гарантирует, что предсказания для позиции i могут зависеть только от выходов в позициях, меньших i.

Внимание

Функция внимания может быть описана как отображение запроса и набора пар ключ-значение в выход, где запрос, ключ и значение – это векторы.
Выход вычисляется как взвешенная сумма значений; вес назначается функцией совместимости запроса с соответствующим ключом.

Внимание как масштабированное скалярное произведение

Частичное внимание вычисляется как скалярное произведение (рис. 7).

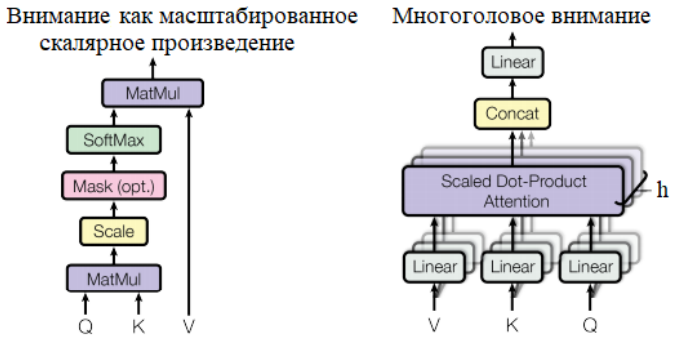

Рис. 7 Два механизма внимания [20]

На входе слоя внимания векторы запросов и ключей размерности d_k и значений размерности d_v. На практике из векторов составляются матрицы, и выходная матрица внимания вычисляется следующим образом:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V,$$

где *Q, K* и *V* – соответственно матрицы запросов, ключей и значений.
При больших d_k скалярное произведение может возрасти, перемещая softmax в область чрезмерно малых градиентов.
Чтобы этому противодействовать, скалярное произведение делится на квадратный корень из d_k.

Многоголовое внимание

Многоголовое внимание – это применение нескольких функций внимания (рис. 7): запросы, ключи и значения отображаются в выход *h* раз, где *h* – число параллельных слоев внимания, или голов.
Получаемые отображения объединяются в результате выполнения операции конкатенации:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O,$$

$$\text{где head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V);$$

$$W_i^Q \in R^{d_{model} \times d_k};\ W_i^K \in R^{d_{model} \times d_k};\ W_i^V \in R^{d_{model} \times d_v};\ W^O \in R^{hd_v \times d_{model}},$$

d_{model} – размер входной последовательности.

В [20] $h = 8$; $_{dk} = d_v = d_{model} / h = 64$.
Уменьшение размерности слоев, реализующих головы внимания, делает временные издержки вычисления внимания такими же, как и при единичном внимании полной размерности.

Применение многоголового внимания

Трансформер по-разному использует многоголовое внимание:
В слоях внимания декодера запросы поступают от предыдущего уровня декодера, а ключи и значения – с выхода кодера.
Это позволяет каждой позиции текущего слоя декодера посещать все позиции во входной последовательности.
Это имитирует типичный механизм внимания кодера-декодера при работе с моделями *последовательность в последовательность*.
Кодер содержит слои самовнимания, в которых все ключи, значения и запросы берутся с выхода предыдущего слоя кодера.
Каждая позиция текущего слоя кодера посещает все предшествующие ей позиции.
Точно так же в декодере слои самовнимания позволяют каждой позиции посещать все предшествующие ей позиции. Чтобы предотвратить в декодере поток информации справа налево и сохранить тем самым авторегрессию, на данные, поступающие на softmax, накладывается маска: данные, соответствующие недопустимым посещениям, заменяются на ?∞.

Нейронная сеть прямого распространения

Полносвязый подслой, имеющийся в каждом блоке кодера и декодера, – это двухслойная полносвязная НС прямого распространения (feed-forward network, FFN) с функцией активации ReLU на первом слое.
НС одинаково обрабатывает каждую позицию последовательности, реализуя следующую функцию:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2,$$

где W_i и b_i – соответственно веса и смещения слоев НС.
Заметим, что такую же функцию реализует НС с двумя одномерными сверточными слоями и единичным ядром свертки.
В [20] размер входа и выхода FFN равен 512, а выход ее первого слоя – 2048.

Эмбединг-слои и softmax

Эмбединг-слои преобразовывают числовые индексы токенов в векторы размера d_{model}. Выход декодера преобразуется в прогноз вероятностей следующего токена обучаемым линейным слоем и softmax. В обоих емебединг-слоях и линейном слое, предшествующем softmax, используется одна и та же матрица весов. В емебединг-слоях ее веса умножаются на корень квадратный из d_{model}.

Кодирование позиций

Поскольку модель не является рекуррентной и сверточной, то для учета порядка следования токенов необходимо предоставить сведения об их относительных или абсолютных позициях в подаваемой на вход модели последовательности.
Для это цели применяется кодирование позиций. Вектор с кодами позиций имеет ту же размерность, что представляющий токен эмбединг-вектор. Это позволяет выполнять суммирование этих двух векторов.

Для кодирования позиции взяты следующие функции:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}});\ PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}),$$

где *pos* – это позиция, *i* – измерение (номер элемента в эмбединге позиций).
Таким образом, каждое значения вектора позиционного кодирования соответствуют синусоиде. Предполагается, что выбранные функции позволят модели легко обучится посещать относительные позиции, поскольку для любого заданного отступа k значение PE_{pos+k} может быть представлено как линейная функция PE_{pos}.
Вдобавок выполнены эксперименты по получения обучаемых эмбедингов позиций, которые показали, что оба подхода показывают сравнимые результаты.
Выбран первый подход, поскольку что он позволяет экстраполировать данные на последовательности, длина которых больше длины последовательностей, встречающихся во время обучения (случай, когда число токенов во входной последовательности, меньше заданного размера входа).

Почему самовнимание

Выбор самовнимания взамен рекуррентных и сверточных моделей выполнен в результате анализа трех следующих показателей: вычислительная сложность, пересчитанная на один слой, возможности для распараллеливания и длина пути в НС при выявлении удаленных зависимостей.
Одним из ключевых факторов, влияющих на выявление удаленных зависимостей, является длина путей прямого и обратного распространения сигналов в НС: чем меньше длина, тем проще поиск зависимостей.
Слой самовнимания обрабатывает все позиции с постоянным числом последовательно выполняемых операций, в то время как рекуррентный слой требует выполнения $O(n)$ последовательных операций, где n – длина последовательности. Таким образом, слой самовнимания быстрее рекуррентного слоя, если длина последовательности меньше представляющего ее вектора, что, в частности, характерно для современных задач машинного перевода. В задачах с длинными последовательностями с целью снижения объема вычислений самовнимание в каждой позиции может быть ограничено посещением заданного числа соседей.
Сверточные слои, если их применять взамен слоев самовнимания, так же в общем случае уступают последним по показателю "длина пути".

Обучение

Модель обучалась машинному переводу с английского на немецкий и с английского на французский.
В первом случае обучение велось на наборе данных WMT 2014, содержащем около 4.5 миллионов пар предложений. Предложения кодировались с употреблением общего токенизатора с размером словаря токенизатора 37000. Во втором число предложений 36М. В словаре 32000 токенов. Каждый обучающий пакет с набором пар предложений содержал примерно 25000 входных и 25000 целевых токенов.
Обучение велось машинах с 8 NVIDIA P100 GPUs. Каждый шаг при обучении базовой модели длился 0.4 с. Всего во время обучения сделано 100'000 шагов (12 часов). При обучении большой модели длительность шага 1.0 с. Число шагов 300'000 (3.5 дня). Алгоритм оптимизации Adam с шагом обучения, зависящим от номера шага. В качестве регуляризаторов использованы Dropout (слой прореживания) с коэффициентом прореживания 0.1 и сглаживание меток.

Заключение

Представленный в [20] трансформер является первой трансдукционной (преобразующей) моделью, полностью полагающейся на самовнимание. В задачах машинного перевода трансформер можно обучить заметно быстрее, чем рекуррентные и сверточные НС. На обоих использованных набора данных достигнуты результаты, превосходящие все известные.
Очевидно, что нет препятствий применению трансформеров для обработки других типов данных, таких, как изображения, аудио и видео.
Код создания, обучения и тестирования трансформера доступен на https://github.com/tensorflow/tensor2tensor.

5.14. GPT

5.14.1. Описание модели

GPT (Generative Pre-training Transformer) [21] – это НС, обученная без учителя НС на большом числе текстов.
Модель GPT, реализованная на архитектуре Трансфомер, является однонаправленной, используя при обучении левый контекст для максимизации правдоподобия $L_1(U)$ корпуса $U = \{u_1, ..., u_n\}$:

$L_1(U) = \Sigma_i \log P(u_i \mid u_{i-k}, ..., u_{i-1}; \theta)$,

где k – размер левого контекста, а условная вероятность P моделируется НС с параметрами θ. Параметры обучаются с использованием стохастического градиентного спуска.
На этапе моделирования ЕЯ используется многослойный декодер на архитектуре Трансфомер [20].
Эта модель с целью передачи декодеру наиболее значимой части информации применяет к входным данным операцию многоголового самовнимания (каждая голова имеет свое мнение, а решение принимается взвешенным голосованием):

$$h_0 = UW_e + W_p;$$

$$h_l = \text{transformer\_block}(h_l - 1)\ \forall\ i \in [1, n];$$

$$P(u) = \text{softmax}(h_n W_e^T),$$

где $U = (u_{-k}, ..., u_{-1})$ – контекстный вектор токенов;
n – число слоев;
W_e – матрица векторных представлений токенов;
W_p – матрица векторных представлений позиций токенов (моделирует, как токен в одной позиции взаимодействует с токеном в другой позиции).
Вероятность моделируется функцией softmax.
После обучения модели с целевой функцией $L_1(U)$ выполняется настройка (дополнительное обучение) модели на решаемую

задачу обработки ЕЯ, например, классификацию документов.
В задаче классификации документов для получения их векторных представлений берется версия gpt2 модели GPT2ForSequenceClassification (GPT-2 является развитием GPT).
Словарь модели версии gpt2 содержит 50257 токенов:

```
from transformers import GPT2Tokenizer
pretrained_name = 'gpt2' # Версия обученной модели
tokenizer = GPT2Tokenizer.from_pretrained(pretrained_name)
vocab = tokenizer.get_vocab()
print(len(vocab)) # 50257
```

5.14.2. Блоки и слои модели

Модель GPT2ForSequenceClassification версии gpt2 содержит 12 блоков следующего вида (на примере блока 0):

```
(0): Block(
  (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  (attn): Attention(
    (c_attn): Conv1D()
    (c_proj): Conv1D()
    (attn_dropout): Dropout(p=0.1, inplace=False)
    (resid_dropout): Dropout(p=0.1, inplace=False)
  )
  (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  (mlp): MLP(
    (c_fc): Conv1D()
    (c_proj): Conv1D()
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

Все слои модели:

```
GPT2ForSequenceClassification(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0): Block(...)
      ...
      (11): Block(...)
    )
    (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (score): Linear(in_features=768, out_features=2, bias=False)
)
```

Код получения сведений о слоях модели:

```
from transformers import GPT2ForSequenceClassification
pretrained_name = 'gpt2' # Версия обученной модели
model = GPT2ForSequenceClassification.from_pretrained(pretrained_name)
print(model)
```

GPT2ForSequenceClassification отличается от базовой модели GPT2Model версии gpt2 score-весами.

5.14.3. Подготовка данных и результаты

Подготовка данных выполняется в 2 этапа: на первом с использованием GPT2ForSequenceClassification версии gpt2 получаются и сохраняются в файлы векторы документов корпуса, на втором векторы документов одновременно с их метками загружаются и передаются на вход классификатору. (Дополнительно берется модель GPT2LMHeadModel так же версии gpt2.)
В программах получения и использования векторов *m_num* = 0.
Векторы токенов берутся с выхода последнего блока модели.
Вектор документа – это усредненный вектор токенов.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
GPT2LMHeadModel				
1	77.68	81.09	99.33	99.61
2	81.47	81.94	99.33	99.55
3	89.51	88.29	99.33	99.66
GPT2ForSequenceClassification				
1	95.54	94.26	99.33	99.49
2	80.80	80.92	99.33	99.55
3	78.79	79.12	99.33	99.61

5.15. BERT

5.15.1. Описание модели

BERT (bidirectional encoder representations from transformers) [22] – это двунаправленная языковая модель, основанная на архитектуре Трансфомер, включающей НС encoder-decoder с механизмом самовнимания [20]. BERT содержит только кодер (encoder).
Векторные представления токенов в BERT формирует кодер. Вектор токена вычисляется с учетом его правого и левого контекстов, чем и определяется двунаправленность модели. Найденные векторы могут быть использованы для решения различных задач программной обработки ЕЯ.
BERT – это модель языка с маской: случайным образом выбираются слова (15% в каждом предложении), которые обрабатываются следующим образом:
- 80% заменяются на маску [MASK];
- 10% заменяются на случайное слово;
- 10% не изменяются.
Модель учится предсказывать только выбранные слова, в отличие, например, от модели word2vec, которая обучается предсказывать все слова документов корпуса.
Дополнительно BERT обучается определять, являются ли два предложения идущими подряд, что важно, например, в такой задаче, как "ответы на вопросы".
Обучение ведется как на положительных, так и отрицательных примерах: в 50% случаях в паре предложений A и B предложение B действительно идет вслед A и получает метку IsNext (положительный пример), а в оставшихся 50% случайным образом берется из корпуса и снабжается меткой NotNext (отрицательный пример).
BERT обучается в два этапа. На первом создаются векторные представления слов, на втором на их основе строятся модели, предназначенные для решения различных прикладных задач, например, "ответы на вопросы", "машинный перевод" или "порождение текста".
На первом этапе на вход BERT подаются последовательности из маскированных предложений и/или пар предложений A и B.
Каждой последовательности предшествует токен [CLS], а в конце предложения A помещается токен [SEP] (рис. 8).

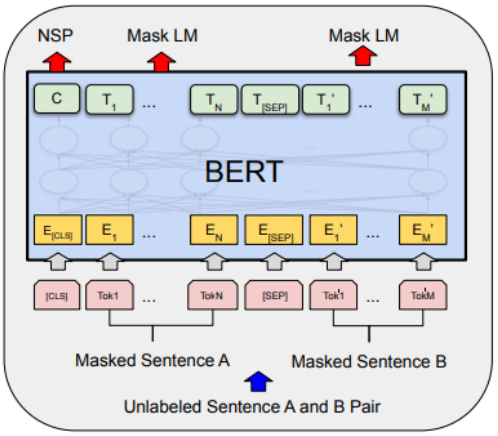

Рис. 8. Схема реализации первого этапа обучения BERT; случай двух предложений на входе [22]

На рис. 8 использованы следующие обозначения:
NSP – next sentence prediction (предсказание следующего предложения);
Mask LM – mask language model (маскированная модель языка);
Mask sentence A (B) – маскированное предложение A (B);
Unlabeled Sentence A and B Pair – пара A и B неразмеченных предложений.
Результатом обучения являются:
- E[CLS], E1, ..., EN, E[SEP], E'1, ..., E'M – входные векторные представления (эмбединги) токенов [CLS], Tok1, ..., TokN, [SEP] , Tok'1, ..., Tok'M предложений A и B;
- скрытый вектор C токена [CLS];
- скрытые векторы T1, ..., TN, T[SEP], T'1, ..., T'M, соответствующие входным токенам Tok1, ..., TokN, [SEP] , Tok'1, ..., Tok'M.
Представление каждого токена формируется как сумма трех эмбедингов: токена, сегмента и позиции.
Первый характеризует токен, второй – принадлежность токена предложению A или B, а третий – позицию токена в предложении (рис. 9).

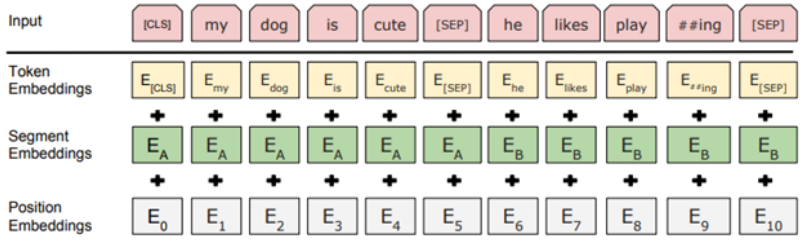

Рис. 9. Представление токена [22]

На рис. 9 использованы следующие обозначения:
Input – вход;
Token Embeddings – эмбединги токенов;
Segment Embeddings – эмбединги сегментов;
Position Embeddings – эмбединги позиций.
Обучение BERT выполнено на BooksCorpus – 800M слов и English Wikipedia – 2,500M слов.
Предобученную модель BERT, обращаясь к методам BertTokenizer, BertModel, можно использовать, в частности, для получения векторных представлений предложений и отдельных токенов. Так, для отдельно взятого предложения следует:

- выполнить его токенизацию, записав токены в список tokenized_d;
- заменить токены их индексами, поместив результат в список indexed_tokens;
- создать из единиц список segments_ids такого же размера, как и список tokenized_d, указав тем самым принадлежность всех токенов первому предложению (сегменту);
- преобразовать списки индексов токенов и сегмента в тензоры;
- загрузить предобученную модель BERT и установить режим оценки – прямого распространения данных;
- подать на вход модели тензоры со списками индексов токенов и сегмента и получить выход модели;
- задать номера токена и блока и получить векторное представление токена на выходе блока.

Пример (запускается в Google colab [23]).

```
!pip install transformers
import torch
from transformers import BertTokenizer, BertModel
pretrained_name = 'DeepPavlov/rubert-base-cased' # 'bert-base-uncased'
# Загрузка токенизатора со словарем из модели pretrained_name
tokenizer = BertTokenizer.from_pretrained(pretrained_name)
d = 'Вот первое предложение для получения токенов.'
bert_d = '[CLS]' + d + '[SEP]'
# Получение токенов
tokenized_d = tokenizer.tokenize(bert_d)
print(tokenized_d)
# ['[CLS]', 'вот', 'первое', 'предложение', 'для', 'получения', 'токе', '##нов', '.', '[SEP]']
print('Размер словаря токенизатора:', len(tokenizer.vocab)) # 119547
# Замена токенов их идексами в словаре
indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_d)
# Печать пар токен-индекс
for tup in zip(tokenized_d, indexed_tokens): print(tup)
# ('[CLS]', 101) ('вот', 19030) ('первое', 13656) ('предложение', 16541) ('для', 2748)
# ('получения', 15839) ('токе', 111601) ('##нов', 2763) ('.', 132) ('[SEP]', 102)
# Фиксируем принадлежность токенов первому предложению (сегменту)
len_t = len(tokenized_d)
segments_ids = [1] * len_t # Список из единиц размера len_t
# Преобразуем списки индексов в списки тензоров
tokens_tensors = torch.tensor([indexed_tokens]) # class 'torch.Tensor'
segments_tensors = torch.tensor([segments_ids])
# tensor([101, 19030, 13656, 16541, 2748, 15839, 111601, 2763, 132, 102])
# tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
# Загружаем предобученную модель
model = BertModel.from_pretrained(pretrained_name, output_hidden_states = True)
# Установливаем режим оценки – прямое распространение данных
model.eval()
with torch.no_grad(): # Получаем результат без указания градиента
    outputs = model(tokens_tensors, segments_tensors) # len(outputs) = 3
    # class 'transformers.modeling_outputs.BaseModelOutputWithPoolingAndCrossAttentions'
    # type(outputs[0 , 1]): class 'torch.Tensor'
    # len(outputs[0, 1]) = 1
    # len(outputs[3]) = 13 - число блоков в модели НС
    hidden_states = outputs[2]
    # class 'tuple' [# states, # batches, # tokens, # features]
# Задаем индекс токена и номер блока и получаем векторное представление токена
batch_i = 0 # Всегда нуль
token_i = 3
n_states = len(hidden_states)
state_i = n_states - 4 # 9
print('Токен:', tokenized_d[token_i])
print('Векторное представление токена', token_i)
vec = hidden_states[state_i][batch_i][token_i]
print(vec)
# Токен: предложение
# Векторное представление токена 3
# tensor([ 8.1701e-01, 9.0365e-02, -2.7332e-01,  3.8611e-01, 5.5761e-01,
#         -1.0167e+00, -2.9008e-01,  6.9003e-01, -4.6187e-01, 1.7163e-02,
#         ...
#          5.8677e-01, 5.9555e-01, -2.8127e-01])
```

В примере использован словарь токенов на русском языке, загружаемый из модели DeepPavlov/rubert-base-cased [24], созданной в лаборатории нейронных сетей и глубокого обучения МФТИ [25].

Замечание. Поскольку на вход модели подается одно предложение, то тензоры сегментов можгут быть опущены:

```
outputs = model(tokens_tensors)
```

вместо

```
outputs = model(tokens_tensors, segments_tensors)
```

Тот же результат можно получить с меньшим кодом:

```
!pip install transformers
import torch
from transformers import BertTokenizer, BertModel
import matplotlib.pyplot as plt
pretrained_name = 'DeepPavlov/rubert-base-cased'
```

```
# Загрузка токенизатора со словарем из модели pretrained_name
tokenizer = BertTokenizer.from_pretrained(pretrained_name)
d = 'Вот первое предложение для получения токенов.'
dt = tokenizer(d, return_tensors = 'pt')
# dt - class 'transformers.tokenization_utils_base.BatchEncoding'
print(dt)
#{'input_ids': tensor([[ 101, 19030, 13656, 16541, 2748, 15839, 111601, 2763, 132, 102]]),
# 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]),
# 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
model = BertModel.from_pretrained(pretrained_name, output_hidden_states = True)
model.eval()
with torch.no_grad(): # Получаем результат без указания градиента
   outputs = model(**dt)
hidden_states = outputs[2]
batch_i = 0 # Всегда нуль
token_i = 3
state_i = len(hidden_states) - 4 # 9
print('Векторное представление токена', token_i)
vec = hidden_states[state_i][batch_i][token_i]
print(vec)
# Токен: предложение
# Векторное представление токена 3
# tensor([ 8.1701e-01, 9.0365e-02, -2.7332e-01, 3.8611e-01, 5.5761e-01,
#         -1.0167e+00, -2.9008e-01, 6.9003e-01, -4.6187e-01, 1.7163e-02,
#         ...
#          5.8677e-01, 5.9555e-01, -2.8127e-01])
```

В последнем коде токенизатор отформатировал исходное предложение, добавив в его начало [CLS], а в конец [SEP], и выполнил все необходимые преобразования полученных токенов, подготовив их к передаче модели.
Заметим, что один и тот же токен в разных предложениях будет в общем случае представлен в BERT-модели разными векторами.
К примеру, если подать на вход модели строку

d = 'Это второе предложение для той же цели.'

То для токена *предложение* получим на выходе последнего блока модели уже совсем другое векторное представление:

```
tensor([ 2.7315e-01, 5.3233e-01, -5.0137e-01, 6.3141e-01, -4.6025e-02,
        -6.5314e-01, -2.0771e-01, 7.4391e-01, -4.8299e-01, -8.5190e-01,
        ...
         3.1107e-01, 1.4090e-01, -2.2861e-01])
```

Всего в использованной модели 13 блоков (len(hidden_states)), и векторы токенов можно взять с выхода любого блока либо с выходов разных блоков, применяя операции усреднения или конкатенации, либо комбинируя эти операции. Вдобавок для представления документов корпуса можно взять векторы токена CLS.
Полученные векторы токенов и документов передаются моделям (для обучения или получения результата), ориентированным на решение различных задач обработки ЕЯ.

Замечание. Имеются предобученные модели CamemBERT [26] и FlauBERT [27] для французского языка и PhoBERT [28] для вьетнамского.

5.15.2. Блоки и слои модели

Модель (используется bert-base-uncased) состоит из 14 блоков. Первый содержит 3 слоя Embedding и слои LayerNorm и Dropout.
В модели этот блок задается следующим образом:

```
(embeddings): BertEmbeddings(
  (word_embeddings): Embedding(119547, 768, padding_idx=0)
  (position_embeddings): Embedding(512, 768)
  (token_type_embeddings): Embedding(2, 768)
  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
```

Далее следуют 12 блоков кодера. Каждый блок задается следующим образом (на примере блока 0):

```
(0): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
```

```
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

Завершает модель блок подвыборки:

```
(pooler): BertPooler(
  (dense): Linear(in_features=768, out_features=768, bias=True)
  (activation): Tanh()
)
```

Всю же модель можно описать так:

```
BertModel(
  (embeddings): BertEmbeddings(...)
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0): BertLayer(...)
      ...
      (11): BertLayer(...)
    )
  )
  (pooler): BertPooler(...)
)
```

Код вывода сведений о слоях модели:

```
from transformers import BertModel
pretrained_name = 'bert-base-uncased' # Версия обученной модели
model = BertModel.from_pretrained(pretrained_name)
print(model)
```

5.15.3. Программа получения векторов документов

Векторы документов получаются с использованием модели BERT в Google Colaboratory – общедоступном облачном сервисе [23], позволяющем создавать на Python свои и использовать библиотечные модели разных классов.
Используются 4 способа получения векторов документов. Способ получения векторов регулируется параметром *b_vec*. При задании *b_vec* = 0 и *b_vec* = 3 в качестве векторов документов берутся соответственно векторы токенов [CLS] и [SEP], при других значениях параметра *b_vec* для получения вектора документа суммируются и усредняются векторы токенов этого документа, взятые с последнего блока (*b_vec* = 1) либо с 4-х последних блоков модели (*b_vec* = 2). Отметим, что в двух последних случаях пропускаются токены [CLS] и [SEP], присутствующие соответственно в начале и конце документа.
Код, формирующий векторы документов:

```
# https://huggingface.co/models
!pip install transformers
from sys import exit
import torch, time
from transformers import BertTokenizer, BertModel
from google.colab import drive
drv = '/content/drive/'
drive.mount(drv)
path = drv + 'My Drive/bert/'
save_vocab = False
nw_max = 400
data_set = 2 # 0 - ru; 1 - reuters; 2 - bbc; 3 - docs; 4 - e_docs; 5 - sentiments
tv = 1 # 0 - train; 1 - test; 3 - весь набор
lst_pre_ds = ['', 'r_', 'b_', 'd_', 'e_', 's_']
pre_ds = lst_pre_ds[data_set]
if tv == 0:
  suf = '_t'
elif tv == 1:
  suf = '_v'
else:
  suf = ''
fn_b = path + pre_ds + 'x' + suf + '.txt'
# 0 - cls-векторы последнего блока; 1 - последний блок; 2 - 4 последних блока
# 3 - sep-векторы последнего блока
b_vec = 0
if b_vec == 0:
    fn_b_vecs = 'b_vecs_cls'
elif b_vec == 1:
  fn_b_vecs = 'b_vecs'
elif b_vec == 2:
  fn_b_vecs = 'b_vecs4'
elif b_vec == 3:
  fn_b_vecs = 'b_vecs_sep'
fn_b_vecs = pre_ds + fn_b_vecs + suf + '.txt'
if data_set == 0:
  pretrained_name = 'DeepPavlov/rubert-base-cased'
```

```
  fn_vocab = 'vocab_rubert.txt'
else:
  pretrained_name = 'bert-base-uncased'
  fn_vocab = 'vocab_bert.txt'
# Сохранение списка в текстовый файл
def add_to_txt_f(lst, fn):
    print('Создан файл', fn, 'с числом строк', len(lst))
    with open(fn, 'w', encoding = 'utf-8') as f:
        for val in lst: f.write((val + '\n') if val.find('\n') == -1 else val)
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True):
  with open(fn, 'r', encoding = encoding) as f:
    lst = f.readlines() # class 'list'
  if no_n:
    lst = [x.replace('\n', '') for x in lst]
  print('Прочитан файл', fn, 'с числом строк', len(lst))
  return lst
tokenizer = BertTokenizer.from_pretrained(pretrained_name) #'bert-base-uncased'
t0 = time.time()
vocab = tokenizer.get_vocab()
if save_vocab:
  lst_vocab = [itm[0] + ' : ' + str(itm[1]) for itm in vocab.items()]
  add_to_txt_f(lst_vocab, path + fn_vocab)
  exit()
lst_txt = read_txt_f(fn_b)
lst_tokenized = []
for d in lst_txt:
  d2 = d.split()
  d = ''
  nw = 0
  for w in d2:
    if vocab.get(w) is not None:
      nw += 1
      d += w + ' '
      if nw == nw_max: break
  d = d.rstrip()
  # class 'transformers.tokenization_utils_base.BatchEncoding'
  dt = tokenizer(d, return_tensors = "pt")
  lst_tokenized.append(dt)
t2 = time.time()
print('Время токенизации:', round(t2 - t0, 2))
model = BertModel.from_pretrained(pretrained_name, output_hidden_states = True)
model.eval()
print('Формирование файла', fn_b_vecs)
print('Модель:', pretrained_name)
t3 = time.time()
print('Время оценки модели bert:', round(t3 - t2, 2))
def find_str_vec(vec):
  str_vec = ''
  for v in vec:
    v = float(v.numpy())
    str_vec += (str(round(v, 4)) + ' ')
  return str_vec.rstrip()
fb = open(path + fn_b_vecs, 'w', encoding = 'utf-8') # Результирующий файл
batch_i = 0
n_vecs = 0
print('Получаем и сохраняем векторы документов')
for dt in lst_tokenized: #
  with torch.no_grad():
    outputs = model(**dt)
  if b_vec == 2:
    hidden_states = outputs.hidden_states
    n_states = len(hidden_states) - 1 # 13 блоков
    sz = hidden_states[n_states].size()
  else:
    last_hidden_states = outputs.last_hidden_state
    sz = last_hidden_states.size()
  sz_1 = sz[1]
  if b_vec == 0: # cls - векторы последнего блока
    token_i = 0 # [CLS]
    vec = last_hidden_states[batch_i][token_i]
    str_vec = find_str_vec(vec)
  elif b_vec == 1: # Последний блок
    vec_sum = torch.zeros(768)
    for token_i in range(1, sz_1 - 1):
      vec = last_hidden_states[batch_i][token_i]
      vec_sum += vec
    vec_sum /= (sz_1 - 2)
    str_vec = find_str_vec(vec_sum)
  elif b_vec == 2: # 4 последних блока
    vec_sum = torch.zeros(768)
    n_k = 4
```

```
    for k in range(n_k):
      n_layer = n_states - k
      h_s = hidden_states[n_layer]
      for token_i in range(1, sz_1 - 1):
        vec = h_s[batch_i][token_i]
        vec_sum += vec
    vec_sum /= (sz_1 - 2) * n_k
    str_vec = find_str_vec(vec_sum)
  elif b_vec == 3: # sep - векторы последнего блока
    token_i = sz_1 - 1 # [SEP]
    vec = last_hidden_states[batch_i][token_i]
    str_vec = find_str_vec(vec)
  fb.write(str_vec + '\n')
  n_vecs += 1
  if n_vecs % 200 == 0:
    print('Число векторов в файле:', n_vecs)
fb.close()
print('Всего векторов в файле:', n_vecs)
t4 = time.time()
print('Время получения выходов блоков модели:', round(t4 - t3, 2))
```

Каждый вектор сохраняется в виде строки, в которой его элементы разделены пробелами:

0.0715 -0.4213 0.0692 0.2045 0.7063 -0.1325 0.1285 ... 0.5337 -0.3924

В каждом векторе 768 элементов.

5.15.4. Подготовка данных и результаты

Подготовка данных выполняется в 2 этапа: на первом с использованием BERT получаются и сохраняются в файлы векторы документов корпуса (отдельно для обучающего и проверочного множеств), на втором векторы документов одновременно с их метками загружаются из файла в списки, которые затем передаются на вход классификаторам.
Поочередно в качестве вектора документа берутся:
- вектор токена [CLS];
- усредненные векторы токенов последнего блока BERT;
- усредненные векторы токенов 4-х последних блоков BERT;
- вектор токена [SEP].
При работе с русскими текстами используется предобученная модель DeepPavlov/rubert-base-cased, а с английскими – bert-base-uncased. Словари обеих моделей содержат токены [CLS] и [SEP]:

```
from transformers import BertTokenizer
pretrained_name = 'DeepPavlov/rubert-base-cased'
pretrained_name2 = 'bert-base-uncased'
tokenizer = BertTokenizer.from_pretrained(pretrained_name) #'rubert-base-cased'
vocab = tokenizer.get_vocab()
print(len(vocab)) # 119'547
print(vocab.get('[CLS]')) # 101
print(vocab.get('[SEP]')) # 102
tokenizer = BertTokenizer.from_pretrained(pretrained_name2) #'bert-base-uncased'
vocab = tokenizer.get_vocab()
print(len(vocab)) # 30'522
print(vocab.get('[CLS]')) # 101
print(vocab.get('[SEP]')) # 102
```

Вдобавок взята модель BertForSequenceClassification так же с bert-base-uncased:

```
from transformers import BertForSequenceClassification as mdl
```

В программах получения и использования векторов *m_num* = 1.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
BERT_CLS				
1	98.21	100.0	98.44	99.04
2	97.77	100.0	98.44	98.87
3	97.77	100.0	98.44	99.61
BERT_1				
1	97.54	99.61	98.88	99.27
2	97.99	99.94	99.11	99.27
3	98.21	100.0	98.88	99.21
BERT_4				
1	98.66	99.94	98.66	99.66
2	97.10	98.09	98.88	99.83
3	97.54	99.55	98.88	98.99
BERT_SEP				
1	97.77	99.44	98.88	99.04
2	97.99	99.10	99.11	99.27
3	97.77	98.99	98.66	98.71
BertForSequenceClassification				
1	96.43	98.26	99.11	98.31

2	98.44	99.21	99.11	98.82
3	97.10	97.81	99.11	98.93

5.16. ALBERT

5.16.1. Описание модели

ALBERT (A Lite BERT) имеет существенно меньше параметров, чем BERT.
В модели использованы два способа снижения числа параметров [29]:
- деление большого словаря матрицы эмбедингов на две небольшие матрицы, что позволяет увеличить размер скрытых слоев без серьезного роста размера словаря эмбедингов;
- совместное использование параметров разными слоями, что препятствует росту параметров при увеличении глубины НС.
Уменьшение числа параметров не влечет, как правило, серьезного снижения качества функционирования модели по сравнению с BERT, но позволяет в 1.7 раза быстрее обучить модель, причем с существенно меньшим расходом оперативной памяти.
Токенизатор ALBERT использует SentencePiece-модель (sp-модель) со словарем фиксированной длины.

5.16.2. Формирование и использование SentencePiece-модели

Sp-модель создается SentencePieceTrainer по корпусу из указанного в параметрах метода train файла, в примере – это b_x.txt:

```
import sentencepiece as spm
t_prm = '--input=b_x.txt --model_prefix=b_sp --vocab_size=15362'
spm.SentencePieceTrainer.train(t_prm)
```

Результатом построения sp-модели со словарем из 15362 токенов будут два файла: b_sp.model и b_sp.vocab.
Фрагмент файла b_sp.vocab, сформированному по файлу b_x.txt:

```
<unk>    0
<s>    0
</s>    0
_the    -2.949
s    -2.9582
_to    -3.67326
_f    -3.99644
ed    -4.01554
_in    -4.06046
_and    -4.07854
ing    -4.10428
d    -4.17368
y    -4.17592
_    -4.38446
```

Sp-модель можно загрузить и использовать для получения токенов или их индексов и выполнять обратные операции, получая куски текста по токенам или их индексам:

```
import sentencepiece as spm
sp = spm.SentencePieceProcessor(model_file = 'b_sp.model')
ids = sp.encode('sport news')
print(ids) # [682, 294]
pc = sp.encode('sport news', out_type = str)
print(pc) # ['_sport', '_news']
ids = sp.encode(['sport news', 'private consumption'], out_type = int)
print(ids) # [[682, 294], [1019, 1669, 2738]]
pc = sp.encode(['sport news', 'private consumption'], out_type = str)
print(pc) # [['_sport', '_news'], ['_private', '_con', 'sumption']]
sen = sp.decode([[682, 294], [1019, 1669, 2738]])
print(sen) # ['sport news', 'private consumption']
sen = sp.decode([['_sport', '_news'], ['_private', '_con', 'sumption']])
print(sen) # ['sport news', 'private consumption']
print(sp.get_piece_size()) # 15362 - размер словаря
print(len(sp)) # 15362
print(sp.id_to_piece(2)) # </s>'
print(sp.id_to_piece([1, 2, 7])) # ['<s>', '</s>', 'ed']
print(sp.piece_to_id('<s>')) # 1
print(sp.piece_to_id(['<s>', '</s>', 'ed'])) # [1, 2, 7]
print(sp['</s>']) # 2
```

Имя файла sp-модели указывается в качестве параметра метода, создающего токенизатор:

```
!pip install transformers
!pip install sentencepiece
from google.colab import drive
drv = '/content/drive/'
drive.mount(drv)
path = drv + 'My Drive/albert/'
from transformers import AlbertTokenizer as tkn
vocab_file = path + 'b_sp.model'
tokenizer = tkn(vocab_file) # Создаем токенизатор
lst_txt = <Формируем список документов>
# Предельное число токенов во входном тензоре; определяется параметрами модели
nw_max = 512
```

```
for d in lst_txt:
  dt = tokenizer(d)['input_ids']
  if len(dt) > nw_max: dt = dt[:nw_max]
  dt = torch.tensor([dt])
  lst_tokenized.append(dt)
```

5.16.3. Слои модели

Слои модели AlbertForSequenceClassification версии albert-base-v2:

```
AlbertForSequenceClassification(
  (albert): AlbertModel(
    (embeddings): AlbertEmbeddings(
      (word_embeddings): Embedding(30000, 128, padding_idx=0)
      (position_embeddings): Embedding(512, 128)
      (token_type_embeddings): Embedding(2, 128)
      (LayerNorm): LayerNorm((128,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0, inplace=False)
    )
    (encoder): AlbertTransformer(
      (embedding_hidden_mapping_in): Linear(in_features=128, out_features=768, bias=True)
      (albert_layer_groups): ModuleList(
        (0): AlbertLayerGroup(
          (albert_layers): ModuleList(
            (0): AlbertLayer(
              (full_layer_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (attention): AlbertAttention(
                (query): Linear(in_features=768, out_features=768, bias=True)
                (key): Linear(in_features=768, out_features=768, bias=True)
                (value): Linear(in_features=768, out_features=768, bias=True)
                (attention_dropout): Dropout(p=0, inplace=False)
                (output_dropout): Dropout(p=0, inplace=False)
                (dense): Linear(in_features=768, out_features=768, bias=True)
                (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              )
              (ffn): Linear(in_features=768, out_features=3072, bias=True)
              (ffn_output): Linear(in_features=3072, out_features=768, bias=True)
              (dropout): Dropout(p=0, inplace=False)
            )
          )
        )
      )
    )
    (pooler): Linear(in_features=768, out_features=768, bias=True)
    (pooler_activation): Tanh()
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

Веса albert-base-v2, не используемые при инициализации модели AlbertForSequenceClassification:
['predictions.bias', 'predictions.LayerNorm.weight', 'predictions.LayerNorm.bias', 'predictions.dense.weight',
'predictions.dense.bias', 'predictions.decoder.weight', 'predictions.decoder.bias']

5.16.4. Подготовка данных и результаты

Подготовка данных, как и в случае BERT, выполняется в 2 этапа: на первом с использованием в ALBERT версии модели albert-base-v2 получаются и сохраняются в файлы векторы документов корпуса, на втором векторы документов одновременно с их метками загружаются и передаются на вход классификатору.
Размер словаря токенизатора: 30000.
В программах получения и использования векторов *m_num* = 2.
Векторы токенов берутся с выхода последнего блока модели. Вектор документа – это усредненный вектор токенов.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
AlbertModel				
1	95.76	96.06	96.43	96.68
2	93.53	97.19	96.43	97.13
3	94.42	97.92	96.43	96.85
AlbertForSequenceClassification				
1	93.75	97.69	97.10	96.90
2	94.20	97.24	96.65	97.36
3	94.20	96.79	96.43	96.96

5.17. RoBERTa

5.17.1. Описание модели

RoBERTa (robustly optimized BERT pretraining approach) [30] имеет ту же архитектуру, что и BERT, улучшая последнюю

модель за счет следующих мероприятий:
- увеличение времени обучения модели и обучающих пакетов;
- устранение целевой функции для предсказания следующего предложения;
- увеличение размеров обучающих последовательностей;
- динамическое изменение маски, применяемой к обучающим данным.
Вдобавок для изучения влияния размера обучающего множества на результат обучения использован новый набор данных CC-NEWS [31], сопоставимый по размеру с другими используемыми наборами.
Предпринятые шаги приводят к росту качества решения типовых задач обработки ЕЯ (ответы на вопросы, определение эквивалентности двух вопросов и пр.).

5.17.2. Блоки и слои модели

RoBERTa – это кодер со слоями внимания.
RobertaForSequenceClassification (версия roberta-base) содержит 12 следующих блоков (на примере блока 0):

```
(0): RobertaLayer(
  (attention): RobertaAttention(
    (self): RobertaSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): RobertaSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): RobertaIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): RobertaOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

Все слои модели:

```
RobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0): RobertaLayer(...)
        ...
        (11): RobertaLayer(...)
      )
    )
  )
  (classifier): RobertaClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

Веса версии roberta-base, не используемые при инициализации RobertaForSequenceClassification:
['lm_head.bias', 'lm_head.dense.weight', 'lm_head.dense.bias', 'lm_head.layer_norm.weight', 'lm_head.layer_norm.bias', 'lm_head.decoder.weight', 'roberta.pooler.dense.weight', 'roberta.pooler.dense.bias']

5.17.3. Подготовка данных и результаты

Берется версия roberta-base модели RobertaForSequenceClassification (*m_num* = 7).
Размер словаря токенизатора: 50265.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
1	98.21	98.59	99.11	99.77
2	96.65	97.81	99.33	99.77
3	98.44	98.76	99.11	99.77

5.18. BertGeneration

5.18.1. Описание модели

BertGeneration [32] – это модель, основанная на архитектуре Трансфомер, для инициализации весов которой могут использованы веса предобученных моделей, таких, как BERT, GPT-2 и RoBERTa.
Использование весов обспечивает повышение качества решения таких задач, как машинный перевод, реферирование текста, деление текста на предложения и составление текста из предложений.
Модель включает кодер и декодер, которые можно объединить е единую модель:

```
from transformers import BertGenerationEncoder as bge, BertGenerationDecoder as bgd
from transformers import EncoderDecoderModel, BertTokenizer
# Формируем Bert2Bert модель на базе известных предобученных моделей
# Используем [CLS]- и [SEP]-токены BERT как BOS и EOS-токены
# BOS и EOS-токены – соответственно токены начала и конца последовательности слов
pretrained_name = 'bert-large-uncased'
encoder = bge.from_pretrained(pretrained_name,
                        bos_token_id = 101, eos_token_id = 102)
# Добавляем слои с перекрестным вниманием
decoder = bgd.from_pretrained(pretrained_name, add_cross_attention = True,
                        is_decoder = True,
                        bos_token_id = 101, eos_token_id = 102)
bert2bert = EncoderDecoderModel(encoder = encoder, decoder = decoder)
# Создаем токенизатор
tokenizer = BertTokenizer.from_pretrained(pretrained_name)
txt = 'This is a long article to summarize'
input_ids = tokenizer(txt, add_special_tokens = False, return_tensors = 'pt').input_ids
labels = tokenizer('This is a short summary', return_tensors = 'pt').input_ids
# Обучение
loss = bert2bert(input_ids = input_ids, decoder_input_ids = labels, labels = labels).loss
loss.backward() # Обратное распространение ошибки
...
gen_ids = bert2bert.generate(input_ids)
gen_txt = tokenizer.decode(gen_ids[0], skip_special_tokens = False)
print(gen_txt)
```

5.18.2. Подготовка данных и результаты

Используем в BertGeneration для получения векторов документов кодер BertGenerationEncoder версии bert-base-uncased и токенизатор BertTokenizer.
Векторы токенов снимаются с последнего блока кодера.
В программах получения и использования векторов $m\_num = 4$.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
1	98.44	99.94	99.33	99.16
2	97.99	99.89	99.33	99.04
3	98.21	99.89	99.33	99.16

5.19. ConvBERT

5.19.1. Описание модели

Модель ConvBERT [33] является модификацией BERT, получаемой в результате использования динамической свертки на основе окна: ядро свертки генерируются путем взятия локального окна текущего токена, что позволяет выявить локальные зависимости токенов и, как следствие, – разные значения совпадающих по написанию токенов (омонимов).

5.19.2. Блоки и слои модели

ConvBertForSequenceClassification версии YituTech/conv-bert-base содержит 12 следующих блоков (на примере блока 0):

```
(0): ConvBertLayer(
  (attention): ConvBertAttention(
    (self): ConvBertSelfAttention(
      (query): Linear(in_features=768, out_features=384, bias=True)
      (key): Linear(in_features=768, out_features=384, bias=True)
      (value): Linear(in_features=768, out_features=384, bias=True)
      (key_conv_attn_layer): SeparableConv1D(
        (depthwise): Conv1d(768, 768, kernel_size=(9,), stride=(1,), padding=(4,), groups=768, bias=False)
        (pointwise): Conv1d(768, 384, kernel_size=(1,), stride=(1,), bias=False)
      )
      (conv_kernel_layer): Linear(in_features=384, out_features=54, bias=True)
      (conv_out_layer): Linear(in_features=768, out_features=384, bias=True)
      (unfold): Unfold(kernel_size=[9, 1], dilation=1, padding=[4, 0], stride=1)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): ConvBertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
```

```
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): ConvBertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): ConvBertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

Все слои модели:

```
ConvBertForSequenceClassification(
  (convbert): ConvBertModel(
    (embeddings): ConvBertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): ConvBertEncoder(
      (layer): ModuleList(
        (0): ConvBertLayer(...)
        ...
        (11): ConvBertLayer(...)
      )
    )
  )
  (classifier): ConvBertClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

В ConvBertForSequenceClassification версии YituTech/conv-bert-base (в отличие от базовой модели ConvBertModel) заново инициализируются следующие веса:
['classifier.out_proj.bias', 'classifier.dense.weight', 'classifier.out_proj.weight', 'classifier.dense.bias'].

5.19.3. Подготовка данных и результаты

Используются ConvBertModel и ConvBertForSequenceClassification версии YituTech/conv-bert-base (*m_num* = 5).
Размер словаря токенизатора: 29514. Векторы снимаются с последнего блока модели.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
ConvBertModel				
1	93.75	94.43	95.31	95.61
2	91.96	91.90	95.76	95.22
3	94.20	94.94	95.31	95.27
ConvBertForSequenceClassification				
1	94.20	95.27	96.21	95.95
2	90.85	91.39	95.54	95.67
3	93.30	94.99	95.31	95.33

5.20. BART

5.20.1. Описание модели

BART [34] реализует оригинальный метод настройки модели для решения задач обработки ЕЯ, следуя приведенной на рис. 10 схеме.

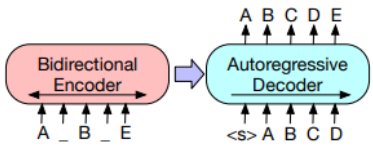

Рис. 10. Схема тонкой настройки (дополнительного обучения) BART [34]

В поступающем на вход двунаправленного кодера документе (рис. 10) фрагменты текста заменены маскирующими символами.
Восстановление документа выполняет авторегрессионный декодер.
Для тонкой настройки модели подлинный документ подается на входы и кодера, и декодера.
Результат снимается с последнего скрытого слоя декодера.

Пример.

```
from transformers import BartForConditionalGeneration as bcg, BartTokenizer
model = bcg.from_pretrained('facebook/bart-large', force_bos_token_to_be_generated = True)
tok = BartTokenizer.from_pretrained('facebook/bart-large')
example_english_phrase = 'UN Chief Says There Is No <mask> in Syria'
batch = tok(example_english_phrase, return_tensors = 'pt')
generated_ids = model.generate(batch['input_ids'])
gen_sen = tok.batch_decode(generated_ids, skip_special_tokens = True)
print(gen_sen) # ['UN Chief Says There Is No Plan to Stop Chemical Weapons in Syria']
```

В словаре токенизатора предобученой модели facebook/bart-base 50265 токенов.

Замечание. Имеется предобученная модель BARThez [35] для французского языка.

5.20.2. Блоки и слои модели

Кодер версии facebook/bart-base содержит 6 следующих блоков (на примере блока 0):

```
(0): BartEncoderLayer(
  (self_attn): BartAttention(
    (k_proj): Linear(in_features=768, out_features=768, bias=True)
    (v_proj): Linear(in_features=768, out_features=768, bias=True)
    (q_proj): Linear(in_features=768, out_features=768, bias=True)
    (out_proj): Linear(in_features=768, out_features=768, bias=True)
  )
  (self_attn_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  (fc1): Linear(in_features=768, out_features=3072, bias=True)
  (fc2): Linear(in_features=3072, out_features=768, bias=True)
  (final_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
)
```

В декодере так же 6 блоков (на примере блока 0):

```
(0): BartDecoderLayer(
  (self_attn): BartAttention(
    (k_proj): Linear(in_features=768, out_features=768, bias=True)
    (v_proj): Linear(in_features=768, out_features=768, bias=True)
    (q_proj): Linear(in_features=768, out_features=768, bias=True)
    (out_proj): Linear(in_features=768, out_features=768, bias=True)
  )
  (self_attn_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  (encoder_attn): BartAttention(
    (k_proj): Linear(in_features=768, out_features=768, bias=True)
    (v_proj): Linear(in_features=768, out_features=768, bias=True)
    (q_proj): Linear(in_features=768, out_features=768, bias=True)
    (out_proj): Linear(in_features=768, out_features=768, bias=True)
  )
  (encoder_attn_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  (fc1): Linear(in_features=768, out_features=3072, bias=True)
  (fc2): Linear(in_features=3072, out_features=768, bias=True)
  (final_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
)
```

Все слои модели:

```
BartModel(
  (shared): Embedding(50265, 768, padding_idx=1)
  (encoder): BartEncoder(
    (embed_tokens): Embedding(50265, 768, padding_idx=1)
    (embed_positions): BartLearnedPositionalEmbedding(1026, 768, padding_idx=1)
    (layers): ModuleList(
      (0): BartEncoderLayer(...)
      (1): BartEncoderLayer(...)
      ...
      (5): BartEncoderLayer(...)
    )
    (layernorm_embedding): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (decoder): BartDecoder(
    (embed_tokens): Embedding(50265, 768, padding_idx=1)
    (embed_positions): BartLearnedPositionalEmbedding(1026, 768, padding_idx=1)
    (layers): ModuleList(
      (0): BartDecoderLayer(...)
      (1): BartDecoderLayer(...)
      ...
    )
    (layernorm_embedding): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
)
```

При инициализации BartForSequenceClassification не используются следующие веса: ['final_logits_bias'].

5.20.3. Подготовка данных и результаты

Берутся BartModel и BartForSequenceClassification (*m_num* = 3) версии facebook/bart-base.
В случае BartModel векторы токенов поочередно снимаются с выхода последнего блока кодера или декодера. В второй модели – только декодера.
Размер словаря токенизатора: 50265.
Вектор документа – это усредненный вектор токенов.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
BartModel. Выход кодера				
1	98.21	99.16	99.11	99.16
2	98.44	99.55	99.55	98.99
3	97.99	99.16	99.33	99.21
BartModel. Выход декодера				
1	98.44	99.27	99.33	99.77
2	97.54	99.16	99.33	99.72
3	96.88	98.42	99.33	99.83
BartForSequenceClassification. Выход декодера				
1	99.11	100.0	99.78	99.72
2	99.33	99.94	99.55	99.72
3	99.33	100.0	99.55	99.83

5.21. DeBERTa

5.21.1. Описание модели

DeBERTa (BERT с улучшенным декодированием и рассеянным вниманием) [36] – улучшает BERT и RoBERTa за счет следующих новаций:
- использование рассеянного внимания, при котором каждый токен представляется векторами его контента и позиции, в результате чего веса внимания к токенам вычисляются с использованием рассеянных матриц их контентов и относительных позиций;
- использование на этапе предварительного обучении модели декодера с расширенной маской для включения абсолютных позиций в слой декодирования с целью прогнозирования маскированных токенов.
Вдобавок для точной настройки применен новый метод виртуального состязательного обучения, улчшающий обобщающие свойства модели.

5.21.2. Блоки и слои модели

DeBERTa – это кодер с элементами внимания.
В DebertaForSequenceClassification (версия microsoft/deberta-base) 12 следующих блоков (на примере блока 0):

```
(0): DebertaLayer(
  (attention): DebertaAttention(
    (self): DisentangledSelfAttention(
      (in_proj): Linear(in_features=768, out_features=2304, bias=False)
      (pos_dropout): StableDropout()
      (pos_proj): Linear(in_features=768, out_features=768, bias=False)
      (pos_q_proj): Linear(in_features=768, out_features=768, bias=True)
      (dropout): StableDropout()
    )
    (output): DebertaSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): DebertaLayerNorm()
      (dropout): StableDropout()
    )
  )
  (intermediate): DebertaIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): DebertaOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): DebertaLayerNorm()
    (dropout): StableDropout()
  )
)
```

Все слои модели:

```
DebertaForSequenceClassification(
  (deberta): DebertaModel(
    (embeddings): DebertaEmbeddings(
      (word_embeddings): Embedding(50265, 768, padding_idx=0)
      (LayerNorm): DebertaLayerNorm()
      (dropout): StableDropout()
    )
    (encoder): DebertaEncoder(
      (layer): ModuleList(
        (0): DebertaLayer(...)
```

```
      ...
      (11): DebertaLayer(...)
    )
    (rel_embeddings): Embedding(1024, 768)
  )
 )
 (pooler): ContextPooler(
  (dense): Linear(in_features=768, out_features=768, bias=True)
  (dropout): StableDropout()
 )
 (classifier): Linear(in_features=768, out_features=2, bias=True)
 (dropout): StableDropout()
)
```

5.21.3. Подготовка данных и результаты

Версия microsoft/deberta-base. Размер словаря токенизатора: 50265, *m_num* = 8.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
DebertaModel				
1	97.99	99.27	99.11	98.71
2	98.21	98.20	99.11	98.87
3	98.44	99.49	98.88	99.21
DebertaForSequenceClassification				
1	97.54	98.93	99.11	99.04
2	98.66	98.54	99.11	99.32
3	98.21	98.48	99.11	99.21

5.22. DistilBERT

5.22.1. Описание модели

DistilBERT [37] – это упрощенная версия BERT [22]. По сравнению с BERT размер модели уменьшен на 40%, скорость функционирования увеличена на 60%, снижение показателей качества – 3%.

5.22.2. Блоки и слои модели

В DistilBERT (версия distilbert-base-uncased) 6 следующих блоков (на примере блока 0):

```
(0): TransformerBlock(
(attention): MultiHeadSelfAttention(
  (dropout): Dropout(p=0.1, inplace=False)
  (q_lin): Linear(in_features=768, out_features=768, bias=True)
  (k_lin): Linear(in_features=768, out_features=768, bias=True)
  (v_lin): Linear(in_features=768, out_features=768, bias=True)
  (out_lin): Linear(in_features=768, out_features=768, bias=True)
)
(sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
(ffn): FFN(
  (dropout): Dropout(p=0.1, inplace=False)
  (lin1): Linear(in_features=768, out_features=3072, bias=True)
  (lin2): Linear(in_features=3072, out_features=768, bias=True)
)
(output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
)
```

Все слои модели:

```
DistilBertModel(
  (embeddings): Embeddings(
    (word_embeddings): Embedding(30522, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (transformer): Transformer(
    (layer): ModuleList(
      (0): TransformerBlock(...)
      ...
      (5): TransformerBlock(...)
    )
  )
)
```

5.22.3. Подготовка данных и результаты

Версия distilbert-base-uncased. Размер словаря токенизатора: 30522, *m_num* = 9.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
DistilBertModel				
1	98.88	99.44	99.55	99.10
2	98.44	99.10	99.55	99.21
3	97.99	98.26	99.33	99.44
DistilBertForSequenceClassification				
1	96.88	98.03	99.33	99.32
2	98.66	99.77	99.33	99.16
3	97.10	98.26	99.55	99.16

5.23. ELECTRA

5.23.1. Описание модели

В ELECTRA [38] реализован иной, по сравнению с BERT, подход к обучению модели. В BERT часть токенов замещаются маской, и модель обучается восстанавливать первичные токены. Такой подход позволяет качественно обучать модель, но требует значительных временных ресурсов. В ELECTRA вместо маски часть токенов замещаются токенами, поставляемыми небольшой сетью-генератором, и в модели сеть-дискриминатор обучается определять, является ли токен замещенным или нет. Такой подход, согласно экспериментам, более эффективен, чем маскирование токенов.

5.23.2. Блоки и слои модели

В ElectraForSequenceClassification (версия google/electra-base-discriminator) 12 следующих блоков (на примере блока 0):

```
(0): ElectraLayer(
  (attention): ElectraAttention(
    (self): ElectraSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): ElectraSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): ElectraIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): ElectraOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

Все слои модели:

```
ElectraForSequenceClassification(
  (electra): ElectraModel(
    (embeddings): ElectraEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): ElectraEncoder(
      (layer): ModuleList(
        (0): ElectraLayer(...)
        ...
        (11): ElectraLayer(...)
      )
    )
  )
  (classifier): ElectraClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

5.23.3. Подготовка данных и результаты

Модель ElectraForSequenceClassification. Версия google/electra-base-discriminator. Размер словаря токенизатора: 30522, *m_num* = 10.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
1	84.15	83.51	95.98	95.16
2	90.85	90.60	95.31	95.67
3	76.56	75.86	95.54	94.60

5.24. Funnel Transformer

5.24.1. Описание модели

В Funnel-Transformer [39] постепенно уменьшается число слоев в последовательно идущих блоках (рис. 11).

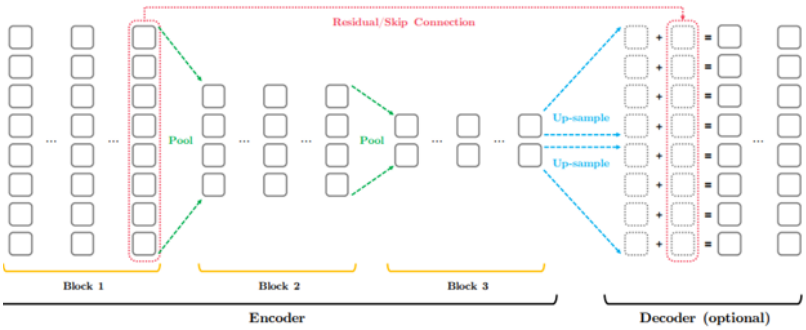

Рис. 11. Архитектура Funnel-Transformer [39]

Сэкономленные ресурсы инвестируются в увеличение глубины модели.

5.24.2. Блоки и слои модели

В FunnelForSequenceClassification (версия funnel-transformer/medium-base) следующие блоки (на примере блока 0):

```
(0): FunnelLayer(
  (attention): FunnelRelMultiheadAttention(
    (hidden_dropout): Dropout(p=0.1, inplace=False)
    (attention_dropout): Dropout(p=0.1, inplace=False)
    (q_head): Linear(in_features=768, out_features=768, bias=False)
    (k_head): Linear(in_features=768, out_features=768, bias=True)
    (v_head): Linear(in_features=768, out_features=768, bias=True)
    (post_proj): Linear(in_features=768, out_features=768, bias=True)
    (layer_norm): LayerNorm((768,), eps=1e-09, elementwise_affine=True)
  )
  (ffn): FunnelPositionwiseFFN(
    (linear_1): Linear(in_features=768, out_features=3072, bias=True)
    (activation_dropout): Dropout(p=0.0, inplace=False)
    (linear_2): Linear(in_features=3072, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (layer_norm): LayerNorm((768,), eps=1e-09, elementwise_affine=True)
  )
)
```

Все слои модели:

```
FunnelForSequenceClassification(
  (funnel): FunnelBaseModel(
    (embeddings): FunnelEmbeddings(
      (word_embeddings): Embedding(30522, 768)
      (layer_norm): LayerNorm((768,), eps=1e-09, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): FunnelEncoder(
      (attention_structure): FunnelAttentionStructure(
        (sin_dropout): Dropout(p=0.1, inplace=False)
        (cos_dropout): Dropout(p=0.1, inplace=False)
      )
      (blocks): ModuleList(
        (0): ModuleList(
          (0): FunnelLayer(...)
          ...
          (5): FunnelLayer(...)
        )
        (1): ModuleList(
          (0): FunnelLayer(...)
          (1): FunnelLayer(...)
          (2): FunnelLayer(...)
        )
        (2): ModuleList(
          (0): FunnelLayer(...)
```

```
        (1): FunnelLayer(...)
        (2): FunnelLayer(...)
       )
      )
     )
    )
   (classifier): FunnelClassificationHead(
     (linear_hidden): Linear(in_features=768, out_features=768, bias=True)
     (dropout): Dropout(p=0.1, inplace=False)
     (linear_out): Linear(in_features=768, out_features=2, bias=True)
   )
)
```

5.24.3. Подготовка данных и результаты

Модель FunnelForSequenceClassification. Версия funnel-transformer/medium-base. Размер словаря токенизатора: 30522, *m_num* = 11.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
1	89.96	88.24	95.98	96.62
2	92.41	93.64	95.76	96.62
3	93.08	94.26	95.31	96.29

5.25. LED и Longformer

5.25.1. Описание модели

Модели на основе архитектуры трансформера со традиционными операциями самовнимания работают с последовательностями длиной 512. Длинные последовательности с 1000 и более токенов не применяются из-за черзмерного роста числа операций самовнимания, которое находится в квадратичной зависимости от длины последовательности.
В LED (Longformer-Encoder-Decoder) и Longformer [40] число операций внимания находится в линейной зависимости от длины последовательности, что позволяет обрабатывать документы с числом токенов более 1000.
В Longformer стандартное самовнимание заменено сочетанием локального (оконного) внимания с глобальным вниманием, мотивированным задачей. Как и в прежних вариантах трансформеров с длинными последовательностями, модель оценивается на уровне символов.
Самовнимание в Longformer реализуется как на локальном, так и глобальном контекстах. Большинство токенов обращаются друг к другу только локально: каждый токен посещает *w*/2 предшествующих и *w*/2 последующих токенов. Размер окна *w* определяется config.attention_window, которое может быть и списком, задающим различные значения окна для каждого слоя.
Несколько выбранных токенов реализуют глобальное внимание, обращаясь ко всем другим токенам, подобно тому, как это может быть задано для всех токенов в BertSelfAttention. Локально и глобально посещающие токены обслуживаются различными матрицами запросов, ключей и значений. Каждый локально посещающий токен посещает токены как в своем окне, так и все глобально посещающие токены, что делает глобальное внимание симметричным.
Пользователь может выделить локально и глобально посещающие токены, задав подходящий тензор global_attention_mask, указав в нем 0 или 1 соответственно для токена посещающего локально или глобально.
Используемый в Longformer механизм самовнимания снижает время и память, необходимые для умножения матриц запросов и ключей с $O(ns \times ns)$ до $O(ns \times w)$, где ns – размер последовательности токенов, а w – средний размер окна, при условии, что глобально посещающих токенов существенно меньше токенов, посещающих локально.

Замечание. Сегменты токенов разделяются символом tokenizer.sep_token (</s>) с кодом 2 в словаре токенов.

5.25.2. Блоки и слои моделей

Кодер LEDForSequenceClassification (версия allenai/led-base-16384) содержит 6 следующих блоков (на примере блока 0):

```
(0): LEDEncoderLayer(
  (self_attn): LEDEncoderAttention(
    (longformer_self_attn): LEDEncoderSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (query_global): Linear(in_features=768, out_features=768, bias=True)
      (key_global): Linear(in_features=768, out_features=768, bias=True)
      (value_global): Linear(in_features=768, out_features=768, bias=True)
    )
    (output): Linear(in_features=768, out_features=768, bias=True)
  )
  (self_attn_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  (fc1): Linear(in_features=768, out_features=3072, bias=True)
  (fc2): Linear(in_features=3072, out_features=768, bias=True)
  (final_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
)
```

Пример блока декодера LEDForSequenceClassification:

```
(0): LEDDecoderLayer(
  (self_attn): LEDDecoderAttention(
    (k_proj): Linear(in_features=768, out_features=768, bias=True)
```

```
    (v_proj): Linear(in_features=768, out_features=768, bias=True)
    (q_proj): Linear(in_features=768, out_features=768, bias=True)
    (out_proj): Linear(in_features=768, out_features=768, bias=True)
   )
   (self_attn_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
   (encoder_attn): LEDDecoderAttention(
    (k_proj): Linear(in_features=768, out_features=768, bias=True)
    (v_proj): Linear(in_features=768, out_features=768, bias=True)
    (q_proj): Linear(in_features=768, out_features=768, bias=True)
    (out_proj): Linear(in_features=768, out_features=768, bias=True)
   )
   (encoder_attn_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
   (fc1): Linear(in_features=768, out_features=3072, bias=True)
   (fc2): Linear(in_features=3072, out_features=768, bias=True)
   (final_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
```

Все слои LEDForSequenceClassification:

```
LEDForSequenceClassification(
  (led): LEDModel(
    (shared): Embedding(50265, 768, padding_idx=1)
    (encoder): LEDEncoder(
      (embed_tokens): Embedding(50265, 768, padding_idx=1)
      (embed_positions): LEDLearnedPositionalEmbedding(16384, 768, padding_idx=1)
      (layers): ModuleList(
        (0): LEDEncoderLayer(...)
        (5): LEDEncoderLayer(...)
      )
      (layernorm_embedding): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    )
    (decoder): LEDDecoder(
      (embed_tokens): Embedding(50265, 768, padding_idx=1)
      (embed_positions): LEDLearnedPositionalEmbedding(1024, 768, padding_idx=1)
      (layers): ModuleList(
        (0): LEDDecoderLayer(...)
        ...
        (5): LEDDecoderLayer(...)
      )
      (layernorm_embedding): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    )
  )
  (classification_head): LEDClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.0, inplace=False)
    (out_proj): Linear(in_features=768, out_features=3, bias=True)
  )
)
```

LongformerForSequenceClassification (версия allenai/longformer-base-4096) содержит 12 следующих блоков (на примере блока 0):

```
(0): LongformerLayer(
  (attention): LongformerAttention(
    (self): LongformerSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (query_global): Linear(in_features=768, out_features=768, bias=True)
      (key_global): Linear(in_features=768, out_features=768, bias=True)
      (value_global): Linear(in_features=768, out_features=768, bias=True)
    )
    (output): LongformerSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): LongformerIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): LongformerOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

Все слои LongformerForSequenceClassification:

```
LongformerForSequenceClassification(
  (longformer): LongformerModel(
    (embeddings): LongformerEmbeddings(
```

```
    (word_embeddings): Embedding(50265, 768, padding_idx=1)
    (position_embeddings): Embedding(4098, 768, padding_idx=1)
    (token_type_embeddings): Embedding(1, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
   )
   (encoder): LongformerEncoder(
    (layer): ModuleList(
     (0): LongformerLayer(...)
     ...
     (11): LongformerLayer(...)
    )
   )
  )
  (classifier): LongformerClassificationHead(
   (dense): Linear(in_features=768, out_features=768, bias=True)
   (dropout): Dropout(p=0.1, inplace=False)
   (out_proj): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

5.25.3. Подготовка данных и результаты

Модель LEDForSequenceClassification; версия allenai/led-base-16384; Размер словаря токенизатора: 50265, *m_num* = 12.
Модель LongformerForSequenceClassification; версия allenai/longformer-base-4096; Размер словаря токенизатора: 50265, *m_num* = 13.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
LEDForSequenceClassification. Выход декодера				
1	98.66	100.0	99.11	99.83
2	98.88	100.0	99.33	99.89
3	98.66	100.0	99.33	99.77
LongformerForSequenceClassification				
1	97.32	97.24	99.55	99.49
2	97.54	97.07	99.55	99.61
3	98.88	97.81	99.55	99.55

5.26. MobileBERT

5.26.1. Описание модели

MobileBERT [41] – это компактная, инвариантная к задачам обработки ЕЯ версия BERT для устройств с ограниченными ресурсами.
Суть инвариантности в том, что предобученная базовая модель затем настраивается на решаемую задачу обработки ЕЯ.
MobileBERT построена по подобию инвертированной BERT_LARGE.

5.26.2. Блоки и слои модели

Кодер MobileBertForSequenceClassification (версия google/mobilebert-uncased) содержит 24 следующих блоков (на примере блока 0):

```
(0): MobileBertLayer(
  (attention): MobileBertAttention(
    (self): MobileBertSelfAttention(
      (query): Linear(in_features=128, out_features=128, bias=True)
      (key): Linear(in_features=128, out_features=128, bias=True)
      (value): Linear(in_features=512, out_features=128, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): MobileBertSelfOutput(
      (dense): Linear(in_features=128, out_features=128, bias=True)
      (LayerNorm): NoNorm()
    )
  )
  (intermediate): MobileBertIntermediate(
    (dense): Linear(in_features=128, out_features=512, bias=True)
  )
  (output): MobileBertOutput(
    (dense): Linear(in_features=512, out_features=128, bias=True)
    (LayerNorm): NoNorm()
    (bottleneck): OutputBottleneck(
      (dense): Linear(in_features=128, out_features=512, bias=True)
      (LayerNorm): NoNorm()
      (dropout): Dropout(p=0.0, inplace=False)
    )
  )
  (bottleneck): Bottleneck(
    (input): BottleneckLayer(
```

```
      (dense): Linear(in_features=512, out_features=128, bias=True)
      (LayerNorm): NoNorm()
    )
    (attention): BottleneckLayer(
      (dense): Linear(in_features=512, out_features=128, bias=True)
      (LayerNorm): NoNorm()
    )
  )
  (ffn): ModuleList(
    (0): FFNLayer(
      (intermediate): MobileBertIntermediate(
        (dense): Linear(in_features=128, out_features=512, bias=True)
      )
      (output): FFNOutput(
        (dense): Linear(in_features=512, out_features=128, bias=True)
        (LayerNorm): NoNorm()
      )
    )
    (1): FFNLayer(
      (intermediate): MobileBertIntermediate(
        (dense): Linear(in_features=128, out_features=512, bias=True)
      )
      (output): FFNOutput(
        (dense): Linear(in_features=512, out_features=128, bias=True)
        (LayerNorm): NoNorm()
      )
    )
    (2): FFNLayer(
      (intermediate): MobileBertIntermediate(
        (dense): Linear(in_features=128, out_features=512, bias=True)
      )
      (output): FFNOutput(
        (dense): Linear(in_features=512, out_features=128, bias=True)
        (LayerNorm): NoNorm()
      )
    )
  )
)
```

Все слои модели:

```
MobileBertForSequenceClassification(
  (mobilebert): MobileBertModel(
    (embeddings): MobileBertEmbeddings(
      (word_embeddings): Embedding(30522, 128, padding_idx=0)
      (position_embeddings): Embedding(512, 512)
      (token_type_embeddings): Embedding(2, 512)
      (embedding_transformation): Linear(in_features=384, out_features=512, bias=True)
      (LayerNorm): NoNorm()
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (encoder): MobileBertEncoder(
      (layer): ModuleList(
        (0): MobileBertLayer(..)
        ...
        (23): MobileBertLayer(...)
      )
    )
    (pooler): MobileBertPooler()
  )
  (dropout): Dropout(p=0.0, inplace=False)
  (classifier): Linear(in_features=512, out_features=2, bias=True)
)
```

5.26.3. Подготовка данных и результаты

Модель MobileBertForSequenceClassification; версия google/mobilebert-uncased; Размер словаря токенизатора: 30522, *m_num* = 14.

Результаты низкие.

5.27. Transfomer XL

5.27.1. Описание модели

Языковые модели на основе трансформеров принимают сегменты текста фиксированной длины, что препятствует выявлению смысловых закономерностей и отношений между словами.
Transfomer XL [42] позволяет выявлять долгосрочные зависимости, выходя за пределы фиксированных сегментов текста. Такую возможность предоставляет механизм сегментно-уровневой возвратности и новая схема кодирования позиций. Кроме того, Transfomer XL решает проблему фрагментации контекста. В Transfomer XL глубина выявления долгосрочных зависимостей превышает на 80% рекуррентные НС и на 450% условного трансформера, который обучается на коротких последовательностях без учета контекстной информации предшествующих последовательностей.

5.27.2. Блоки и слои модели

TransfoXLForSequenceClassification (версия transfo-xl-wt103) содержит 18 следующих блоков (на примере блока 0):

```
(0): RelPartialLearnableDecoderLayer(
  (dec_attn): RelPartialLearnableMultiHeadAttn(
    (qkv_net): Linear(in_features=1024, out_features=3072, bias=False)
    (drop): Dropout(p=0.1, inplace=False)
    (dropatt): Dropout(p=0.0, inplace=False)
    (o_net): Linear(in_features=1024, out_features=1024, bias=False)
    (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    (r_net): Linear(in_features=1024, out_features=1024, bias=False)
  )
  (pos_ff): PositionwiseFF(
    (CoreNet): Sequential(
      (0): Linear(in_features=1024, out_features=4096, bias=True)
      (1): ReLU(inplace=True)
      (2): Dropout(p=0.1, inplace=False)
      (3): Linear(in_features=4096, out_features=1024, bias=True)
      (4): Dropout(p=0.1, inplace=False)
    )
    (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  )
)
```

Все слои модели:

```
TransfoXLForSequenceClassification(
  (transformer): TransfoXLModel(
    (word_emb): AdaptiveEmbedding(
      (emb_layers): ModuleList(
        (0): Embedding(20000, 1024)
        (1): Embedding(20000, 256)
        (2): Embedding(160000, 64)
        (3): Embedding(67735, 16)
      )
      (emb_projs): ParameterList(
          (0): Parameter containing: [torch.FloatTensor of size 1024x1024]
          (1): Parameter containing: [torch.FloatTensor of size 1024x256]
          (2): Parameter containing: [torch.FloatTensor of size 1024x64]
          (3): Parameter containing: [torch.FloatTensor of size 1024x16]
      )
    )
    (drop): Dropout(p=0.1, inplace=False)
    (layers): ModuleList(
      (0): RelPartialLearnableDecoderLayer(...)
      ...
      (17): RelPartialLearnableDecoderLayer(...)
    )
    (pos_emb): PositionalEmbedding()
  )
  (score): Linear(in_features=1024, out_features=2, bias=False)
)
```

Веса версии squeezebert/squeezebert-uncased, не используемые при инициализации TransfoXLForSequenceClassification:
['crit.cluster_weight', 'crit.cluster_bias', 'crit.out_layers.0.weight', 'crit.out_layers.0.bias', 'crit.out_layers.1.weight',
'crit.out_layers.1.bias', 'crit.out_layers.2.weight', 'crit.out_layers.2.bias', 'crit.out_layers.3.weight', 'crit.out_layers.3.bias',
'crit.out_projs.0', 'crit.out_projs.1', 'crit.out_projs.2', 'crit.out_projs.3']

5.27.3. Подготовка данных и результаты

TransfoXLForSequenceClassification, версия squeezebert/squeezebert-uncased, Размер словаря токенизатора: 267'735, *m_num* = -20.

Результаты не получены ввиду исчерпания ресурсов.

5.28. XLNet

5.28.1. Описание модели

BERT [22] успешно применяет в задачах обработки ЕЯ модель с маской, не выявляя, однако, зависимостей между предсказанными токенами (предсказываются токены, замещенные маской).
XLNet [43], пытаясь преодолеть этот недостаток, использует обобщенный авторегрессионный метод обучения с двунаправленным контекстом с перестановками токенов (при обучении для предсказания токена x_t берутся различные перестановки токенов его контекстного окна).
Авторегрессионные модели (АМ) языка при заданной последовательности токенов $x = (x_1, \dots, x_T)$ максимизируют правдоподобие корпуса, выраженное произведением

$p(x) = \Pi^{T}_{t=1}\, p(x_t \mid \mathbf{x}_{<t})$.

или при обратном обходе последовательности токенов

$p(x) = \Pi^{1}_{t=T}\, p(x_t \mid \mathbf{x}_{>t})$.

АМ обучаются кодировать однонаправленный контекст (левый или правый), в то время как задачи понимания ЕЯ требуют привлечение двунаправленного контекста.
Автокодировщики (АК) проектируются для восстановления испорченных данных, например, BERT при создании предобученной модели часть токенов замещает маской [MASK] и обучается восстанавливать исходные токены.
При тонкой настройке модели на конкретную задачу ЕЯ маска отсутствует, что влечет конфликт между предобучением и тонкой настройкой. Кроме того, токены, замещенные маской, рассматриваются моделью как независимые, что не всегда соответствует действительности.
XLNet использует обобщенную авторегрессию, объединяющую АМ и АК. XLNet максимизирует ожидаемый логарифм правдоподобия последовательности, представленной всеми перестановками токенов, в результате чего контекст каждой позиции может включать как левые, так и правые токены. При этом XLNet, в отличие от BERT, не искажает данные.
Вдобавок XLNet при предобучении интегрирует механизм повторения сегментов и относительное кодирование, применное в Transformer-XL [42].
Следует заметить, что кодирование позиций выполняется по реальной последовательности токенов, а перестановки реализуются за счет изменения порядка ее факторизации, что достигается в результате применения подходящей маски внимания.

5.28.2. Блоки и слои модели

XLNetForSequenceClassification (версия xlnet-base-cased) содержит 11 следующих блоков (на примере блока 1):

```
(1): XLNetLayer(
  (rel_attn): XLNetRelativeAttention(
    (layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (ff): XLNetFeedForward(
    (layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (layer_1): Linear(in_features=768, out_features=3072, bias=True)
    (layer_2): Linear(in_features=3072, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (dropout): Dropout(p=0.1, inplace=False)
)
```

Все слои модели:

```
XLNetForSequenceClassification(
  (transformer): XLNetModel(
    (word_embedding): Embedding(32000, 768)
    (layer): ModuleList(
      (1): XLNetLayer(...)
      ...
      (11): XLNetLayer(...)
    )
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (sequence_summary): SequenceSummary(
    (summary): Linear(in_features=768, out_features=768, bias=True)
    (first_dropout): Identity()
    (last_dropout): Dropout(p=0.1, inplace=False)
  )
  (logits_proj): Linear(in_features=768, out_features=2, bias=True)
)
```

```
XLNetForSequenceClassification не использует следующие веса:
xlnet-base-cased: ['lm_loss.weight', 'lm_loss.bias'].
```

5.28.3. Подготовка данных и результаты

XLNetForSequenceClassification, версия xlnet-base-cased, Размер словаря токенизатора: 32000, *m_num* = 21.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
1	97.77	97.75	99.11	99.55
2	90.62	91.67	99.55	99.72
3	97.54	98.31	99.55	99.77

5.29. MPNet

5.29.1. Описание модели

BERT [22] успешно применяет в задачах обработки ЕЯ модель с маской. При этом, однако, теряется возможность выявить позиционные зависимости между токенами, замещенными маской.
XLNet [43], пытаясь преодолеть этот недостаток, использует перестановочную модель (см. разд. 5.28.1). Применение такой модели не позволяет, однако, полноценно употребить информацию о позициях токенов, что приводит к дисбалансу позиций в начальной предобученной модели и настраиваемой на ее основе целевой модели.
MPNet [44] пытается нивелировать недостатки, присущие BERT и XLNet: MPNet выявляет зависимости между токенами через перестановочную модель и использует на входе дополнительную информацию о позициях токенов с тем, чтобы

видеть все предложение.

5.29.2. Блоки и слои модели

Кодер MPNetForSequenceClassification (версия google/mobilebert-uncased) содержит 12 следующих блоков (на примере блока 0):

```
(0): MPNetLayer(
  (attention): MPNetAttention(
    (attn): MPNetSelfAttention(
      (q): Linear(in_features=768, out_features=768, bias=True)
      (k): Linear(in_features=768, out_features=768, bias=True)
      (v): Linear(in_features=768, out_features=768, bias=True)
      (o): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (intermediate): MPNetIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): MPNetOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

Все слои модели:

```
MPNetForSequenceClassification(
  (mpnet): MPNetModel(
    (embeddings): MPNetEmbeddings(
      (word_embeddings): Embedding(30527, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): MPNetEncoder(
      (layer): ModuleList(
        (0): MPNetLayer(...)
        ...
        (11): MPNetLayer(...)
      )
      (relative_attention_bias): Embedding(32, 12)
    )
  )
  (classifier): MPNetClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

5.29.3. Подготовка данных и результаты

Модель MPNetForSequenceClassification, версия microsoft/mpnet-base, Размер словаря токенизатора: 30527, *m_num* = 15.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
1	97.77	98.14	99.78	99.72
2	98.88	99.32	99.55	99.72
3	97.90	98.71	99.55	99.66

5.30. SqueezeBert

5.30.1. Описание модели

В SqueezeBert [45] некоторые операции в слоях самовнимания выполняются сгруппированными сверточными слоями, что снижает время отклика НС по сравнению с НС на основе BERT. Например, на Pixel 3 смартфонах время отклика НС снижается в 4.3 раза.
При этом точность результата при проверке модели на наборах данных GLUE [17] сопоставима с BERT [22].

5.30.2. Блоки и слои модели

SqueezeBertForSequenceClassification (версия squeezebert/squeezebert-uncased) содержит 12 следующих блоков (на примере блока 0):

```
(0): SqueezeBertModule(
```

```
  (attention): SqueezeBertSelfAttention(
    (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (dropout): Dropout(p=0.1, inplace=False)
    (softmax): Softmax(dim=-1)
    (matmul_qk): MatMulWrapper()
    (matmul_qkv): MatMulWrapper()
  )
  (post_attention): ConvDropoutLayerNorm(
    (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
    (layernorm): SqueezeBertLayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (intermediate): ConvActivation(
    (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
  )
  (output): ConvDropoutLayerNorm(
    (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
    (layernorm): SqueezeBertLayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

Все слои модели:

```
SqueezeBertForSequenceClassification(
  (transformer): SqueezeBertModel(
    (embeddings): SqueezeBertEmbeddings(
      (word_embeddings): Embedding(30528, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): SqueezeBertEncoder(
      (layers): ModuleList(
        (0): SqueezeBertModule(...)
        ...
        (11): SqueezeBertModule(...)
      )
    )
    (pooler): SqueezeBertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

Веса версии squeezebert/squeezebert-uncased, не используемые при инициализации SqueezeBertForSequenceClassification:
['cls.predictions.bias', 'cls.predictions.transform.dense.weight', 'cls.predictions.transform.dense.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias']

5.30.3. Подготовка данных и результаты

SqueezeBertForSequenceClassification, версия squeezebert/squeezebert-uncased, Размер словаря токенизатора: 30528, *m_num* = 18.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
1	98.88	99.44	99.33	99.10
2	98.66	99.44	99.33	99.27
3	99.11	99.44	99.33	99.16

5.31. T5

5.31.1. Описание модели

T5 (Text-to-Text Transfer Transformer) [46] – это модель кодера-декодера, предварительно обученная с учителем и без него на большом числе задач, в которых данные предварительно преобразуется в формат "текст в текст".
Основная идея T5 – это рассматривать задачи обработки ЕЯ как проблемы "текст в текст", подавая на вход модели и снимая с ее выхода текст.
Работа ведется с моделью t5-base. В словаре токенизатора модели 32100 токенов.

5.31.2. Блоки и слои модели

Кодер t5-base содержит 12 следующих блоков (на примере блока 0):

```
(0): T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=768, out_features=768, bias=False)
        (k): Linear(in_features=768, out_features=768, bias=False)
        (v): Linear(in_features=768, out_features=768, bias=False)
        (o): Linear(in_features=768, out_features=768, bias=False)
        (relative_attention_bias): Embedding(32, 12)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerFF(
      (DenseReluDense): T5DenseReluDense(
        (wi): Linear(in_features=768, out_features=3072, bias=False)
        (wo): Linear(in_features=3072, out_features=768, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
```

Декодер t5-base содержит 12 следующих блоков (на примере блока 0):

```
(0): T5Block(
  (layer): ModuleList(
    (0): T5LayerSelfAttention(
      (SelfAttention): T5Attention(
        (q): Linear(in_features=768, out_features=768, bias=False)
        (k): Linear(in_features=768, out_features=768, bias=False)
        (v): Linear(in_features=768, out_features=768, bias=False)
        (o): Linear(in_features=768, out_features=768, bias=False)
        (relative_attention_bias): Embedding(32, 12)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): T5LayerCrossAttention(
      (EncDecAttention): T5Attention(
        (q): Linear(in_features=768, out_features=768, bias=False)
        (k): Linear(in_features=768, out_features=768, bias=False)
        (v): Linear(in_features=768, out_features=768, bias=False)
        (o): Linear(in_features=768, out_features=768, bias=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (2): T5LayerFF(
      (DenseReluDense): T5DenseReluDense(
        (wi): Linear(in_features=768, out_features=3072, bias=False)
        (wo): Linear(in_features=3072, out_features=768, bias=False)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (layer_norm): T5LayerNorm()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)
```

Все слои модели:

```
T5Model(
  (shared): Embedding(32128, 768)
  (encoder): T5Stack(
    (embed_tokens): Embedding(32128, 768)
    (block): ModuleList(
      (0): T5Block(...)
      ...
      (11): T5Block(...)
    )
    (final_layer_norm): T5LayerNorm()
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (decoder): T5Stack(
    (embed_tokens): Embedding(32128, 768)
    (block): ModuleList(
      (0): T5Block(...)
      (11): T5Block(...)
    )
    (final_layer_norm): T5LayerNorm()
```

```
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
```

5.31.3. Подготовка данных и результаты

Версия t5-base, размер словаря равен 32128, *m_num* = 6. Векторы токенов поочередно берутся с выхода последних блоков кодера и декодера.
Вектор документа – это усредненный вектор токенов.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
T5EncoderModel. Векторы взяты с выхода кодера				
1	99.11	99.77	99.33	99.55
2	98.88	99.89	99.11	99.49
3	98.66	99.83	99.11	99.38
T5Model. Векторы взяты с выхода декодера				
1	95.76	95.50	99.11	96.85
2	94.87	94.43	98.88	96.79
3	83.71	84.36	98.88	96.45

5.32. XLM-Roberta

5.32.1. Описание модели

XLM-Roberta [47] – это модель Roberta, обученная на сотне языков.

5.32.2. Блоки и слои модели

XLMRobertaForSequenceClassification (версия xlm-roberta-base):

```
XLMRobertaForSequenceClassification(
  (roberta): RobertaModel(
    то же, что RobertaForSequenceClassification
  )
)
```

Веса xlm-roberta-base, не используемые при инициализации модели XLMRobertaForSequenceClassification:
['lm_head.bias', 'lm_head.dense.weight', 'lm_head.dense.bias', 'lm_head.layer_norm.weight', 'lm_head.layer_norm.bias', 'lm_head.decoder.weight', 'roberta.pooler.dense.weight', 'roberta.pooler.dense.bias']

5.32.3. Подготовка данных и результаты

XLMRobertaForSequenceClassification, версия xlm-roberta-base, Размер словаря токенизатора: 250'002, *m_num* = 20.

Результаты:

Номер попытки	SGD		HC	
	val_acc	*acc*	*val_acc*	*acc*
1	92.19	92.29	98.44	98.37
2	83.71	85.20	98.21	98.48
3	87.05	86.89	98.21	98.59

6. Сводная таблица результатов

Результаты по всем использованным для классификации документов набора BBCD моделям текста приведены в табл. 2.
Модель оценивается по показателю *vm* = min(*val_acc*$_{HC}$, *acc*$_{HC}$), где *val_acc*$_{HC}$ и *acc*$_{HC}$ – это точности классификации, выполненной НС, соответственно на обучающем и проверочном множествах; значение каждого показателя – это результат усреднения трех измерений (см. разделы с описаниями моделей).
Значения, полученные в результате применения SGDClassifier, приведены в табл. 2 для справки, для оценки моделей текста они не используются, поскольку в большинстве случаев уступают результатам, полученным на НС.
Данные отсортированы по убыванию *vm*.

Таблица 2. Результаты по всем моделям

Модель	*vm*	SGD		HC	
		val_acc	*acc*	*val_acc*	*acc*
BART	99.63	99.26	99.98	99.63	99.76
MPNet	99.63	98.18	98.72	99.63	99.70
Longformer	99.55	97.91	97.37	99.55	99.55
XLNet	99.40	95.31	95.91	99.40	99.68
GPT2	99.33	85.04	84.77	99.33	99.55
LED	99.26	98.73	100.0	99.26	99.83
DistilBERT	99.21	97.55	98.69	99.40	99.21
One-hot	99.18	88.77	87.64	99.18	99.96
RoBERTa	99.18	97.77	98.39	99.18	99.77
SqueezeBERT	99.18	98.88	99.44	99.33	99.18
T5	99.18	98.88	99.83	99.18	99.47

BertGeneration	99.12	98.21	99.91	99.33	99.12
DeBERTa	99.11	98.14	98.65	99.11	99.19
BERT	98.96	97.91	99.85	98.96	99.25
CountVectorizer	98.88	97.10	100.0	98.88	100.0
TfidfVectorizer	98.81	98.88	100.0	98.81	100.0
word2vec	98.36	97.39	99.55	98.36	98.41
XLM-Roberta	98.29	87.65	88.13	98.29	98.48
doc2vec	97.99	96.13	100.0	97.99	99.66
GloVe	97.84	97.32	98.05	97.92	97.84
fasttext	96.77	97.10	98.84	97.62	96.77
ALBERT	96.73	94.05	97.24	96.73	97.07
CB	96.06	92.67	99.85	96.06	96.57
Funnel	95.68	91.82	92.05	95.68	96.51
ConvBERT	95.65	92.78	93.88	95.69	95.65
ELECTRA	95.14	83.85	83.32	95.61	95.14
ЧКМ	95.09	-	-	-	-
LDA	68.53	-	-	-	-
КЧС	29.09	29.32	28.65	23.90	23.90

7. Применение моделей текста для классификации документов разных наборов данных

7.1. Наборы данных

Некоторые модели текста опробованы на 4-х следующих наборах данных, подготовленных для обучения классификаторов документов:
- набор данных для классификации документов (НКД). В НКД [48] 3'261 документа на русском языке, взятых из различных интернет-ресурсов. Число классов 13;
- набор новостных данных Би-би-си (BBCD) [3] содержит 2'225 документов новостного сайта Би-би-си. Число классов 5;
- business articles classification dataset (BACD) [49] содержит 4'167 документов новостных сайтов Globe и PR Newswire. Число классов 8;
- large movie review dataset (LMRD) [50] содержит 50'000 обзоров кинофильмов – положительных и отрицательных. Число классов 2.

В табл. 3 указаны статистические характеристики наборов данных.

Таблица 3. Статистические характеристики наборов данных.

НД	Количество						Размер словаря
	классов	докумен-тов	слов	слогов	многослож-ных слов	букв	
НКД	13	3'261	350'898	1'008'514	106'068	2'391'152	51'356
BBCD	5	2'225	859'442	1'579'770	71'560	3'989'442	27'880
BACD	8	4'167	613'935	1'351'249	98'365	3'387'711	30'854
LMRD	2	50'000	6'598'296	11'840'883	459'753	29'712'104	79'721

Замечание. Многосложным считается слово с числом слогов более 3.

Сложность НД (корпуса) в задаче построения модели текста иллюстрирует отношение V / N, а в задаче и классификации документов – $C * V / D$, где V – размер словаря набора, N – число слов в наборе; C – число классов; D – число документов.
Чем меньше эти показатели, тем чаще (в среднем) каждое слово встречается в корпусе и документах и, следовательно, больше информации для обнаружения связей между словами корпуса и для различения документов.
Другой показатель сложности НД – это точность разметки его документов.
Например, документ из класса *здоровье* может содержать сведения, позволяющие отнести документ к классам *здоровье, спорт* и *семья*. В документах LMRD могут быть одновременно и *положительные*, и *отрицательные*, и *нейтральные* характеристики кинофильмов.
Оценить качество разметки НД в задаче классификации можно, имея обученный классификатор, показывающий хорошие результаты на разных НД, сравнивая точность классификации на новом НД с известными результатами на прежних: относительно низкая точностью указывает на проблемы в разметке документов свежего набора.
Сложность НД снижается при наличии в его классах мало различимых документов. Такими, например, могут быть статьи одного автора на одну и ту же тему.
В табл. 4 приведены оценки сложности НД; в последнем столбце табл. 4 указано значение *v_max* – наибольшей полученной в эксперименте с моделями текста точности классификации документов на оценочном множестве набора.

Таблица 4. Оценки сложности НД

НД	*C*	*D*	*N*	*V*	*V / N*	*C*V / D*	*v_max*
НКД	13	3'261	350'898	51'356	0.1464	204.73	76.05
BBCD	5	2'225	332'357	18'683	0.0562	41.98	99.63
BACD	8	4'167	613'935	30'854	0.0503	59.24	88.60
LMRD	2	50'000	6'598'296	79'721	0.0121	3.19	84.66

Несколько наборов данных, различающихся числом и размеров классов, позволят посмотреть, как модели текста работают в разных условиях.

7.2. Результаты на SGDClassifier

В табл. 5 приведены результаты обучения SGDClassifier, описанного в разд. 3.
Классификатор обучается на каждом наборе с каждой моделью текста трижды.
После обучения замеряются точности классификации на проверочном и обучающем множествах.
В табл. 4 по результатам трех обучений указаны:
val_acc – точность классификации на проверочном множестве;

err – погрешность измерения точности классификации на проверочном множестве;
acc – точность классификации на обучающем множестве.
Кроме того, использованы следующие обозначения моделей:
CV – CountVectorizer;
Tf-idf – TfidfVectorizer;
BERT_1 – BERT, векторы документов взяты с выхода последнего блока модели;
BERT_4 – BERT, векторы документов – это усредненные векторы с выходов 4-х последних блоков модели;
BERT_CLS – BERT, векторы документов – это векторы токена [CLS];
BERT_SEP – BERT, векторы документов – это векторы токена [SEP];
КЧС – вектор кодов слова и его частей;
СВ – случайный вектор;
ЧКМ – частотно-классовая модель.

Таблица 5. Результаты обучения и тестирования SGDClassifier

Модель	Результаты	*val_acc*	*err*	*acc*
НКД. Средняя ошибка *val_acc* без СВ: 1.28%				
one-hot	'0.4947/0.6982;0.411/0.5661;0.5236/0.7654',	47.64	4.36	67.66
CV	'0.656/1.0;0.6347/1.0;0.6454/1.0',	64.54	0.71	100.0
Tf-idf	'0.6986/0.9996;0.7017/0.9996;0.7047/0.9996',	70.17	0.20	99.96
word2vec	'0.6484/0.8771;0.6804/0.8706;0.6743/0.8714',	66.77	1.29	87.30
doc2vec	'0.6454/0.9028;0.6423/0.9124;0.6408/0.9017',	64.28	0.17	90.56
fasttext	'0.6271/0.8771;0.6195/0.8502;0.6377/0.8368',	62.81	0.64	85.47
BERT_1	'0.7032/0.9812;0.6895/0.9869;0.7215/0.9877',	70.47	1.12	98.53
BERT_4	'0.6773/0.9597;0.7093/0.9593;0.7032/0.967',	69.66	1.29	96.20
CB	'0.4049/0.6141;0.3364/0.606;0.3988/0.6248'	38.00	2.91	61.50
fasttext2	'0.2116/0.2189;0.1355/0.1114;0.2374/0.2331'	19.48	-	18.78
LDA	'0.14/0.1809;0.2131/0.2277;0.2466/0.207'	19.99	-	20.52
ЧКМ_1	'63.01/99.65;63.01/99.65;63.01/99.65'	63.01	-	99.65
ЧКМ_2	'0.6591/0.9981;0.656/0.9981;0.6149/0.9988'	64.33	-	99.83
BERT_CLS	'0.6971/0.937;0.6986/0.919;0.6651/0.9159'	68.69	-	92.40
BERT_SEP	'0.6621/0.972;0.6545/0.9351;0.6758/0.9647'	66.41	-	95.73
КЧС	'0.1302/0.9977;0.1302/1.0;0.1271/0.9996'	12.92	-	99.91
word2vec(1)	'0.6073/1.0;0.5967/1.0;0.6088/1.0 (sum_mix = 1)	60.43	-	100.0
word2vec(2)	'0.5799/1.0;0.5693/1.0;0.5693/1.0 (sum_mix = 2)	57.28	-	100.0
BBCD. Средняя ошибка *val_acc* без СВ: 0.68%				
one-hot	'0.8661/0.861;0.9174/0.879;0.8795/0.8891',	88.77	1.98	87.64
CV	'0.9688/1.0;0.9799/1.0;0.9643/1.0',	97.10	0.59	100.0
Tf-idf	'0.9888/1.0;0.9911/1.0;0.9866/1.0',	98.88	0.15	100.0
word2vec	'0.9821/0.9991;0.9576/0.9803;0.9799/0.9893',	97.32	1.04	98.96
doc2vec	'0.9509/0.9977;0.9509/0.9932;0.9598/0.9977',	95.39	0.40	99.62
fasttext	'0.6271/0.8771;0.6195/0.8502;0.6377/0.8368',	97.47	0.20	99.25
GloVe	'0.9665/0.9792;0.9866/0.9837;0.9665/0.9786',	97.32	0.89	98.05
BERT_1	'0.9754/0.9961;0.9799/0.9994;0.9821/1.0',	97.91	0.25	99.85
BERT_4	'0.9866/0.9994;0.971/0.9809;0.9754/0.9955',	97.77	0.60	99.19
CB	'0.9454/0.9994;0.904/0.9983;0.9308/0.9977'	92.67	1.52	99.85
fasttext2	'0.2478/0.238;0.2612/0.2459;0.221/0.2583'	24.33	-	24.74
LDA	'0.5781/0.5701;0.5/0.5734;0.4933/0.5656'	52.38	-	56.97
ЧКМ_2	'0.971/0.9989;0.9777/0.9989;0.9754/0.9994'	97.47	-	99.91
BERT_CLS	'0.9821/1.0;0.9777/1.0;0.9777/1.0',	97.92	-	100.0
BERT_SEP	'0.9777/0.9944;0.9799/0.991;0.9777/0.9899',	97.84	-	99.18
КЧС	'0.2478/0.2459;0.3036/0.3067;0.3212/0.327'	29.09	-	29.32
word2vec(1)	'0.9576/1.0;0.9643/1/0;0.9598/1.0 (sum_mix = 1)	96.06	-	100.0
word2vec(2)	'0.9442/1.0; 0.9308/1.0;0.9375/1.0 (sum_mix = 2)	93.75	-	100.0
BACD. Средняя ошибка *val_acc* без СВ: 1.21%				
one-hot	'0.805/0.8352;0.8218/0.8556;0.7057/0.704',	77.75	4.79	79.83
CV	'0.8577/0.9982;0.8768/0.9982;0.8529/0.9979',	86.25	0.96	99.81
Tf-idf	'0.872/0.9982;0.8672/0.9976;0.866/0.9982',	86.84	0.24	99.80
word2vec	'0.8385/0.8535;0.8481/0.9051;0.8696/0.9096',	85.21	1.17	88.94
doc2vec	'0.8565/0.9337;0.86/0.9364;0.8457/0.9528',	85.41	0.56	94.10
fasttext	'0.8385/0.8604;0.8565/0.8946;0.8624/0.9084',	85.25	0.93	88.78
GloVe	'0.8421/0.8676;0.8278/0.8298;0.8481/0.8565',	83.93	0.77	85.13
BERT_1	'0.8457/0.9349;0.86/0.9499;0.8313/0.9267',	84.57	0.96	93.72
BERT_4	'0.8565/0.9451;0.8481/0.9367;0.8648/0.964',	85.65	0.56	94.86
CB	'0.756/0.837;0.7536/0.8271;0.7787/0.8166'	76.28	1.06	82.69
fasttext2	'0.7751/0.7526;0.7249/0.7238;0.7129/0.7043'	73.76	-	72.69
LDA	'0.6495/0.6046;0.6938/0.6358;0.6878/0.6473'	67.70	-	62.92
ЧКМ_1	'85.05/97.99;85.05/97.99;85.05/97.99'	85.05	-	97.99
ЧКМ_2	'0.8445/0.9922;0.8457/0.9928;0.8313/0.9862'	84.05	-	99.04
BERT_CLS	'0.7859/0.9225;0.8182/0.964;0.8098/0.9628'	80.46	-	94.98
BERT_SEP	'0.8325/0.8709;0.8242/0.8835;0.8481/0.9147'	83.49	-	88.97
КЧС	'0.5063/0.6235;0.4997/0.5546;0.5073/0.6031'	50.44	-	59.37
LMRD. Средняя ошибка *al_acc* без СВ: 0.52%				
one-hot	'0.69/0.6934; 0.6693/0.6727;0.6895/0.6915',	68.29	0.91	68.59
CV	'0.8173/0.9964;0.8059/0.9877;0.8071/0.9908',	81.01	0.48	99.16
Tf-idf	'0.8509/0.9206;0.8511/0.9206;0.8513/0.9205',	85.11	0.01	92.06
word2vec	'0.8204/0.8298;0.8115/0.8196;0.8229/0.8297',	81.83	0.45	82.64
doc2vec	'0.8161/0.8243;0.8201/0.8258;0.8164/0.8266',	81.75	0.17	82.56

fasttext	'0.8171/0.8263;0.82/0.8258;0.8205/0.8258',	81.92	0.14	82.60
GloVe	'0.7932/0.7997;0.7885/0.7932;0.7919/0.8013',	79.12	0.18	79.81
BERT_1	'0.8482/0.8482;0.8456/0.8629;0.849/0.8658',	84.76	0.13	85.90
BERT_4	'0.7988/0.8105;0.8455/0.8638;0.8504/0.8701',	83.16	2.18	84.81
CB	'0.6785/0.6851;0.67/0.6813;0.6774/0.6851'	67.53	0.35	68.38
fasttext2	'0.8146/0.8201;0.8129/0.818;0.8126/0.8192'	81.34	-	81.91
LDA	'0.6172/0.6256;0.6177/0.625;0.613/0.6232'	61.60	-	62.46
ЧКМ_1	'78.4/91.58;78.4/91.58;78.4/91.58'	78.40	-	91.58
ЧКМ_2	'0.8052/0.922;0.7835/0.9153;0.8016/0.9242'	79.68	-	92.05
BERT_CLS	'0.8297/0.8488;0.8328/0.8504;0.8313/0.8483'	83.13	-	84.92
BERT_SEP	'0.8386/0.8519;0.8171/0.8273;0.8321/0.8462'	82.93	-	84.18
КЧС	'0.5063/0.6235;0.4997/0.5546;0.5073/0.6031'	50.44	-	59.37

7.3. Результаты на НС

В табл. 6 приведены результаты обучения НС, описанной в разд. 3.
НС обучается на каждом наборе с каждой моделью текста трижды.
После обучения замеряются точности классификации на проверочном и обучающем множествах.
Использованы те же обозначения, что и в табл. 5.

Таблица 6. Результаты обучения и тестирования НС

Модель	**Результаты**	***val_acc***	***err***	***acc***
НКД. Средняя ошибка *val_acc* без CB: 0.57%				
one-hot	'0.7047/1.0;0.7199/0.9996;0.7306/1.0',	71.84	0.91	99.99
CV	'0.7093/1.0;0.6941/1.0;0.6956/1.0',	69.97	0.64	100.0
Tf-idf	'0.7139/1.0;0.7017/1.0;0.7032/1.0',	70.63	0.51	100.0
word2vec	'0.7139/0.7873;0.7093/0.7231;0.7154/0.7734',	71.29	0.24	76.13
doc2vec	'0.7428/0.8306;0.7321/0.7988;0.7336/0.8322',	73.62	0.44	82.05
fasttext	'0.7184/0.7385;0.7062/0.7262;0.7093/0.7631',	71.13	0.47	74.26
BERT_1	'0.7595/0.9071;0.7702/0.8172;0.7443/0.8848',	75.80	0.91	86.97
BERT_4	'0.7595/0.8944;0.7549/0.8537;0.7671/0.8402',	76.05	0.44	86.28
CB	'0.347/0.5392;0.3881/0.4777;0.4323/0.5376'	38.91	2.88	51.82
fasttext2	'0.1781/0.1528;0.1872/0.1636;0.1811/0.1601'	18.21	-	15.88
LDA_1	'23.29/17.4;23.29/17.4;23.29/17.4'	23.29	-	17.40
LDA_2	'0.344/0.2711;0.3653/0.2688;0.3364/0.2692'	34.86	-	26.97
ЧКМ_1	'63.32/99.65;63.32/99.65;63.32/99.65'	63.32	-	99.65
ЧКМ_2	'0.6773/0.6985;0.6682/0.6816;0.6591/0.6989'	66.82	-	69.30
BERT_CLS	'0.7565/0.8714;0.7534/0.8867;0.7610/0.8637'	75.70	-	87.39
BERT_SEP	'0.7275/0.8518;0.7336/0.8548;0.7352/0.8479'	73.21	-	85.15
КЧС	'0.1669/0.5142;0.173/0.5337;0.1394/0.7224'	15.98	-	59.01
BBCD. Средняя ошибка *val_acc* без CB: 0.14%				
one-hot	'0.9933/0.9994;0.9911/.99940;0.9911/1.0',	99.18	0.10	99.96
CV	'0.9911/1.0;0.9866/1.0;0.9888/1.0',	98.88	0.15	100.0
Tf-idf	'0.9866/1.0;0.9888/1.0;0.9888/1.0',	98.81	0.10	100.0
word2vec	'0.9777/0.9662;0.9777/0.9662;0.9777/0.9657',	97.77	0.00	96.6
doc2vec	'0.9710/0.9426;0.9621/0.9432;0.9688/0.9533',	96.73	0.35	94.64
fasttext	'0.9732/0.9685;0.9799/0.964;0.9754/0.9707',	97.02	0.25	96.57
GloVe	'0.9777/0.9786;0.9799/0.9781;0.9799/0.9786',	97.92	0.10	97.84
BERT_1	'0.9888/0.9927;0.9911/0.9927;0.9888/0.9921',	98.96	0.10	99.25
BERT_4	'0.9866/0.9966;0.9888/0.9983;0.9888/0.9899',	98.81	0.10	99.49
CB	'0.9598/0.9769;0.9621/0.9707;0.9598/0.96'	96.06	0.10	96.92
fasttext2	'0.4196/0.2358;0.2679/0.2150;0.2946/0.2279'	32.74	-	22.62
LDA_1	68.53/71.86	68.53	-	71.86
LDA_2	'0.6272/0.5059;0.6875/0.502;0.692/0.5115'	66.89	-	50.65
ЧКМ_1	95.09/98.42	95.09	-	98.42
ЧКМ_2	'0.9732/0.6927;0.9621/0.7203;0.9710/0.7113'	96.88	-	70.81
BERT_CLS	'0.9844/0.9904;0.9844/0.9887;0.9844/0.9961',	98.44	-	99.17
BERT_SEP	'0.9888/0.9904;0.9911/0.9927;0.9866/0.9871',	98.88	-	99.01
КЧС	'0.2299/0.2465;0.3192/0.2347;0.3103/0.2358'	28.65	-	23.90
BACD. Средняя ошибка *val_acc* без CB: 0.21%				
one-hot	'0.8911/0.9901;0.8876/0.9856;0.8816/0.9802',	88.68	0.34	98.53
CV	'0.8756/0.9967;0.8756/0.9956;0.8816/0.9976',	87.76	0.27	99.66
Tf-idf	'0.8756/0.9973;0.8732/0.9964;0.8744/0.997',	87.44	0.08	99.69
word2vec	'0.8732/0.8334;0.8768/0.858;0.8684/0.8553',	87.28	0.29	84.89
doc2vec	'0.89/0.8976;0.8852/0.8712;0.8828/0.897',	88.60	0.27	88.86
fasttext	'0.8756/0.84;0.872/0.8292;0.872/0.8319',	87.32	0.16	83.37
GloVe	'0.8612/ 0.8319;0.8612/0.8331;0.86/0.8283',	86.08	0.05	83.11
BERT_1	'0.8648/0.861;0.872/0.8994;0.8708/0.8892',	86.92	0.29	88.32
BERT_4	'0.8804/0.8817;0.878/0.9;0.8768/0.8931',	87.84	0.13	89.16
CB	'0.7895/0.7649;0.7955/0.7679;0.7823/0.7649'	78.91	0.45	76.59
fasttext2	'0.7931/0.7295;0.7907/0.7238;0.7943/0.7286'	79.27	-	72.73
LDA_1	'62.8/57.04;62.8/57.04;62.8/57.04'	62.80	-	57.04
LDA_2	'0.7093/0.5206;0.7045/0.5296;0.7033/0.5242'	70.57	-	52.48
ЧКМ_1	'84.93/97.99;84.93/97.99;84.93/97.99'	84.93	-	97.99
ЧКМ_2	'0.8517/0.677;0.86/0.6962;0.8541/0.6998'	85.53	-	69.10
BERT_CLS	'0.8469/0.8805;0.8421/0.9219;0.8457/0.8964'	84.49	-	89.96
BERT_SEP	'0.8708/0.8703;0.8672/0.8718/;0.8684/0.8649'	86.88	-	86.90

КЧС	'0.382/0.7371;0.3261/0.4656;0.3357/0.5608'	34.79	-	58.78
LMRD. Средняя ошибка *al_acc* без CB: 0.14%				
one-hot	'0.8302/0.8702;0.8305/0.856;0.8306/0.867',	83.04	0.02	86.44
CV	'0.7999/0.8561;0.8142/0.8576;0.8114/0.8588',	80.85	0.57	85.75
Tf-idf	'0.8160/0.8636;0.8191/0.8644;0.816/0.8617',	81.70	0.14	86.32
word2vec	'0.8047/0.777;0.8079/0.7712;0.8054/0.7723',	80.60	0.13	77.35
doc2vec	'0.8255/0.807;0.8249/0.8039;0.8247/0.8001',	82.50	0.03	80.37
fasttext	'0.7879/0.7556;0.7905/0.7489;0.7894/0.7499',	78.93	0.09	75.15
GloVe	'0.7784/0.7549;0.7756/0.7526;0.7786/0.7514',	77.75	0.13	75.30
BERT_1	'0.8441/0.8531;0.8427/0.8563;0.8451/0.8536',	84.40	0.08	85.43
BERT_4	'0.8456/0.8614;0.8478/0.8629;0.8465/0.864',	84.66	0.08	86.28
CB	'0.6806/0.6844;0.6721/0.6922;0.6815/0.6933'	67.81	0.40	69.00
fasttext2	'0.7943/0.7556;0.7948/0.7584;0.7933/0.7508'	79.41	-	75.49
LDA_1	'60.95/61.94;60.95/61.94;60.95/61.94'	60.95	-	61.94
LDA_2	'0.6166/0.5983;0.6/0.6065;0.6091/0.6055'	60.86	-	60.34
ЧКМ_1	'78.4/91.58;78.4/91.58;78.4/91.58'	78.40	-	91.58
ЧКМ_2	'0.4999/0.8761;0.4997/0.885;0.4999/0.8791'	49.98	-	88.01
BERT_CLS	'0.8221/0.8442;0.8204/0.8458;0.8271/0.8308'	82.32	-	84.03
BERT_SEP	'0.8277/0.8221;0.8256/0.8303;0.8264/0.8301'	82.66	-	82.75
КЧС	'0.5289/0.5722;0.5028/0.5036;0.5023/0.5022'	51.13	-	52.60

7.4. Вычисление точности и погрешности классификации

Вычисление точности и погрешности классификации по данным табл. 5 и 6 и выполняет следующий код:

```
all_res = [
'0.8267/0.8885;0.8305/0.856;0.8306/0.867',
'0.7999/0.8561;0.7857/0.8611;0.7888/0.8568',
'0.7966/0.8628;0.7985/0.8621;0.7986/0.8612',
'0.8047/0.777;0.8079/0.7712;0.8054/0.7723',
'0.8255/0.807;0.8249/0.8039;0.8247/0.8001',
'0.7879/0.7556;0.7905/0.7489;0.7894/0.7499',
'0.779/0.7513;0.7788/0.749;0.7761/0.7511',
'0.8441/0.8531;0.8427/0.8563;0.8451/0.8536',
'0.8456/0.8614;0.8478/0.8629;0.8465/0.864',
'0.6806/0.6844;0.6721/0.6922;0.6815/0.6933'
]
err = err_avg = 0
for res in all_res:
    res = res.split(';')
    v_acc = acc = 0
    for va in res:
        va = va.split('/')
        v_acc += float(va[0])
        acc += float(va[1])
    v_acc /= 3
    acc /= 3
    err = 0
    for va in res:
        va = va.split('/')
        err += abs(v_acc - float(va[0]))
    err /= len(res)
    err_avg += err
    print(round(100 * v_acc, 2), round(100 * err, 2), round(100 * acc, 2))
err_avg -= err
print(str(round(100 * err_avg / (len(all_res) - 1), 2)) + '%')
```

7.5. Диаграмма сравнительной эффективности моделей текста

На рис. 12 приведена диаграммы сравнительной эффективности моделей текста на всех НД; классификатор НС.
Сравнительная эффективность модели текста на НД – это отношение среднего значения *val_acc* к максимальной величине *val_acc* по всем моделям.
На рис. 12 использованы следующие сокращенные обозначения моделей текста: B_1, B_4, OH, TF, d2v, CV, GV, w2v, FT и CB соответственно для BERT_1, BERT_4, one-hot, Tf-idf, doc2vec, CountVectorizer, GloVe, word2vec, fasttext и случайный вектор.

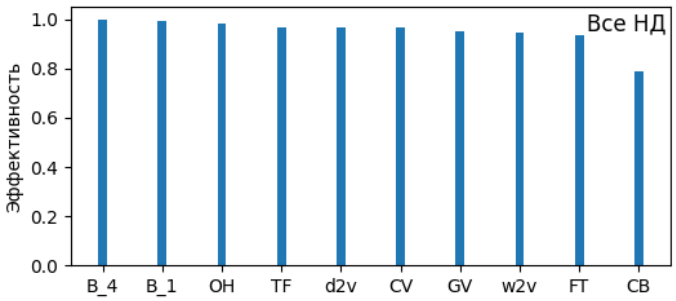

Рис. 12. Диаграмма сравнительной эффективности моделей текста

Диаграмма сравнительной эффективности моделей текста на наборе данных (на всех наборах, если *data_set* = 6) строится

следующим кодом:

```
import numpy as np
import matplotlib.pyplot as plt
data_set = 6  # 6 - все НД
all_r = [[(71.84, 99.99), (69.97, 100.0), (70.63, 100.0), (71.29, 76.13),
         (73.62, 82.05), (71.13, 74.26), (75.80, 86.97), (76.05, 86.28),
         (38.91, 51.82)],
        [],
        [(99.18, 99.96), (98.88, 100.0), (98.81, 100.0), (97.77, 96.6),
         (96.73, 94.64), (97.02, 96.57), (97.77, 96.68), (98.96, 99.25),
         (98.81, 99.49), (96.06, 96.92)],
        [],
        [(88.68, 98.53), (87.76, 99.66), (87.44, 99.69), (87.28, 84.89),
         (88.60, 88.86), (87.32, 83.37), (86.08 , 83.11), (86.92, 88.32),
         (87.84, 89.16), (78.91, 76.59)],
        [(83.04, 86.44), (80.85, 85.75), (81.70, 86.32), (80.60, 77.35),
         (82.50, 80.37), (78.93, 75.15), (77.75, 75.30), (84.40, 85.43),
         (84.66, 86.28), (67.81, 69.00)]]
if data_set in [0, 2, 4, 5]:
    all_res = all_r[data_set]
    all_res = [min(res) for res in all_res]
    max_v = max(all_res)
    all_res = [v / max_v for v in all_res]
elif data_set == 6:  # Все НД
    ins = True
    ind_i = 6
    len_r = len(all_r[2])
    all_res = np.zeros(len_r)
    n_res = 0
    for a_res in all_r:
##        if ins:
##            ins = False
##            continue
        if len(a_res) == 0: continue
        n_res += 1
        b_res = [min(res) for res in a_res]
        max_v = max(b_res)
        b_res = [v / max_v for v in b_res]
        if ins:
            b_res.insert(ind_i, 0)
            ins = False
        c_res = [round(v, 2) for v in b_res]
        print(c_res)
        all_res += np.array(b_res)
    all_res = [res / n_res for res in all_res]
    all_res[6] = 0.95
else:
    print('Плохой номер набора данных')
    exit()
if data_set == 0:
    lst_tm = ['OH', 'CV', 'TF', 'w2v', 'd2v', 'FT',
              'B_1', 'B_4', 'CB']
else:
    lst_tm = ['OH', 'CV', 'TF', 'w2v', 'd2v', 'FT', 'GV',
              'B_1', 'B_4', 'CB']
lst_ds = ['НКД', '', 'BBCD', '', 'BACD', 'LMRD', 'Все НД']
ttl = lst_ds[data_set]
lst_res = []
for tm, res in zip(lst_tm, all_res):
    lst_res.append([tm, res])
lst_res.sort(key = lambda r:(r[1]), reverse = True)
print(lst_res)
accF, accV = [], []
for res in lst_res:
    accF.append(res[0])
    accV.append(res[1])
ind = np.arange(len(lst_res))
width = 0.15 # Ширина столбца диаграммы
plt.figure(figsize = (6, 2.6))
#min_v = min(accV)
#plt.ylim(0.8 * min_v, 1.05) # Пределы по Y
plt.ylim(0, 1.05) # Пределы по Y
plt.bar(ind, accV, width)
plt.xlabel('Модель текста')
plt.ylabel('Эффективность')
plt.title(ttl)
plt.xticks(ind, accF)
plt.show()
```

Заключение

Приведенные в табл. 2 результаты не отражают возможности моделей, прежде всего, трансформеров в решаемой задаче классификации документов, поскольку для трансформеров, кроме использованных базовых версий, имеются и более мощные. В то же время по полученным результатам можно сопоставить модели и сделать вывод о превосходстве трансформеров над прочими (из числа рассмотренных) моделями.
Также следует добавить, что имеются и иные модели текстов, например [51], ELMo (Embeddings from Language Models), Flair, CoVe (Contextual Word Vectors), CVT (Cross-View Training) и пр.

Приложение. Программа подготовки данных и классификации документов

Весь код программы формирования обучающих и проверочных множеств и классификации документов:

```
# !pip install --upgrade gensim # Colab
# Версия питона
# import sys
# ver = sys.version_info.major, sys.version_info.minor
# Предобученные модели (очень много)
# http://vectors.nlpl.eu/repository/
# Функции оценки, потерь, оптимизации
# https://id-lab.ru/posts/developers/funkcii/
# Метрики
# https://www.machinelearningmastery.ru/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226/
# https://en.wikipedia.org/wiki/Precision_and_recall#Precision
# Предобученные модели
# https://medium.com/analytics-vidhya/8-pretrained-models-to-learn-natural-language-processing-nlp-5f14d82e9621
# Набор данных
# http://www.mechanoid.kiev.ua/ml-text-proc.html
# Обзор современных моделей текста
# https://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html#t5
# https://pypi.org/project/fasttext/#text-classification-model
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
# https://github.com/guilhermeeuzebio/bag-of-words
# https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html
# https://nlp.stanford.edu/pubs/glove.pdf - glove
# https://arxiv.org/pdf/1607.04606.pdf - fasttext
# https://arxiv.org/pdf/2009.13658.pdf - position embeddings
# https://habr.com/ru/post/436878/ - bert
# https://arxiv.org/pdf/1810.04805.pdf - bert
# https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/ - bert
# https://huggingface.co/transformers/model_doc/albert.html - albert
# https://www.chrismccormick.ai/offers/nstHFTrM/checkout?utm_source=blog&utm_medium=banner&utm_campaign=ebook_ad&
utm_content=post2 - bert
# https://arxiv.org/pdf/1706.03762.pdf - Attention Is All You Need
# https://www.mql5.com/ru/articles/8909 - самовнимание; трансфомер
# http://www.davidsbatista.net/blog/2020/01/25/Attention-seq2seq/ - внимание
# https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
# https://habr.com/ru/post/341240/ - Transformer
# Transformer-XL:
# https://www.machinelearningmastery.ru/transformer-xl-explained-combining-transformers-and-rnns-into-a-state-of-the-art-
language-model-c0cfe9e5a924/
# https://zen.yandex.ru/media/id/5e048b1b2b616900b081f1d9/intuiciia-za-arhitekturoi-transformatorov-v-nlp-
5fdf48ca8ae4867dadf7101f
# https://towardsdatascience.com/how-to-code-the-transformer-in-pytorch-24db27c8f9ec
# https://huggingface.co/transformers/model_doc/bart.html - bart (seq2seq)
# https://arxiv.org/pdf/1511.06388.pdf - sense2vec
# https://arxiv.org/pdf/1802.05365.pdf - ELMo
# https://huggingface.co/transformers/model_doc/gpt2.html - GPT-2
# ELMo - GPT - BERT
# https://medium.com/@gauravghati/comparison-between-bert-gpt-2-and-elmo-9ad140cd1cda
# https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)
# Latent Semantic Analysis using Python (LSA)
# https://www.datacamp.com/community/tutorials/discovering-hidden-topics-python
# https://webdevblog.ru/podhody-lemmatizacii-s-primerami-v-python/ - spaCy
# Пример классификации
# https://scikit-learn.org/stable/auto_examples/applications/plot_out_of_core_classification.html
# Осочкин А. А., Фомин В. В., Флегонтов А.В. Метод частотно-морфологической классификации текстов.
https://cyberleninka.ru/article/n/metod-chastotno-morfologicheskoy-klassifikatsii-tekstov
# Математические модели текста. http://lab314.brsu.by/kmp-lite/kmp2/JOB/CModel/BoW-Q.htm
# https://habr.com/ru/company/ods/blog/326418/#label-encoding
from sys import exit
##import keras
##print(keras.__version__)
##exit()

import re, numpy as np
import time
colab = not True
if colab:
    from google.colab import drive
    drv = '/content/drive/'
    drive.mount(drv)
```

```
    path = drv + 'My Drive/bert/'
else:
    path = ''
# -1 - создаем корпус - текстовый файл corp0.txt.
#     Перед текстом имя класса, например, '#культура '
# 1 - соокращаем корпус: не менее nw_in_doc_min и не более nw_in_doc_max слов в документе - строке corp2.txt.
# 2 - предварительная обработка текста.
#     Создание набора данных (файлы x.txt и y.txt).
#     Формирование словаря (dict.txt)
#     Если reuters (data_set = 1), то по кодам формируется файл со словами
#     Если bbc (data_set = 2), то данные формируются по файлам в папках директории bbc
# Используем data_set: 0, 2, 4 и 5
# 3 - создание wor2vec / doc2vec / fasttext-модели (параметр vec_model)
#     fasttext2 см. в п. 5
# 4 - загрузка / формирование обучающего и проверочного множеств.
#     Обучение модели (классификатора). Оценка его точности
# 5 - fasttext_2. Создание и самостоятельное применение (model.test)
# 6 - сокращаем reuters, оставляя классы с числом документов, более 100.
#     Сортируем data_set = 0 по классам
#
# step: управляем, меняя vec_model; см. также step = 5
step = 4
data_set = 2 # 0 - ru; 1 - reuters; 2 - bbc; 3 - docs; 4 - e_docs; 5 - sentiments
if data_set not in [0, 2, 4, 5]:
    print('Плохой номер набора данных')
    exit()
#
lst_pre_ds = ['', 'r_', 'b_', 'd_', 'e_', 's_']
pre_ds = lst_pre_ds[data_set]
lem = not True # Лемматизация, если True
sg, size, window, min_cnt, n_iter = 0, 768, 3, 1, 100
if step not in [-1, 1, 2, 3, 4, 5, 6]:
    print('Плохой номер шага')
    exit()
nw_in_doc_max = 150 # Максимальное число слов в документе
nw_in_doc_min = 15 # Минимальное число слов в документе
if step == -1:
    db_name = 'corp.sqlite'
elif step == 2:
    k_split = 0.5 if data_set == 5 else 0.2 # Доля проверочного множества
elif step in [3, 4]:
    vec_model = 50
    # vec_model:
    # 100 - wor2vec
    # 101 - wor2vec из текстового файла (предобученная)
    # 200 - doc2vec
    # 20 - fasttext
    # 30 - fasttext2
    # 40 - GloVe
    # 50 - LDA
    # 50 - Latent Dirichlet Allocation with online variational Bayes algorithm
    #     https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html
    #     https://scikit-learn.org/stable/modules/decomposition.html#latentdirichletallocation
    #     https://jmlr.org/papers/volume3/blei03a/blei03a.pdf
    # 60 - КЧС - модель слова в виде кодов слова и его частей
    # 7 - one-hot
    # 8 - CV - CountVectorizer
    # 9 - Tfidf - TfidfVectorizer
    # 10 - CB - случайно генерируемые векторы слов
    # BERT. Векторы загружаются из b_vecs-файлов
    # 11 - cls-векторы последнего блока
    # 12 - последний блок
    # 13 - 4 последних блока (среднее)
    # 14 - sep-векторы последнего блока
    # 15 - ЧКМ - частотно-классовая модель. Размер вектора слова - число классов.
    #     fi - частота слова в классе i
    #     fi = число присутствий в классе / число присутствий в корпусе
    if vec_model not in [7, 8, 9, 10, 11, 12, 13, 14, 15,
                         20, 30, 40, 50, 60, 100, 101, 200]:
        print('Плохой номер модели')
        exit()
    # Делим набор до получения координат векторов (True)
    # В противном случае получаем список координат векторов и делим этот список
    split_first = True
    k_split = 0.5 if data_set == 5 else 0.2 # Доля проверочного множества
    if step == 4:
        # 0 - усреднение; 1 - усреднение и конкатенация; 2 - конкатенация
        sum_mix = 0
        nv_in_doc_mix = 5 # Число векторов в документе (для конкатенации)
        use_nn = True # Флаг применения нейронной сети в качестве классификатора
        if use_nn:
            # 0 - dense; 1 - +conv1; 2 - +conv2; 3 - +GRU; 4 - +LSTM; 5 - +dense
```

```
            use_conv = 0
            loss = 'categorical_crossentropy' # mse
            epochs = 30 if sum_mix == 2 else 90
        else:
            loss = 'hinge' # hinge, squared_hinge, huber - для SGD
            # При vec_model = 9 (проверить) data_set: 0/2/4/5
            # Лучшие при НС: 76.05/99.33/88.60/85.11
            # 0 - SGD: 70.47/98.88/86.48
            # 1 - AdaBoost: 27.25
            # 2 - RandomForest: 69.41
            # 3 - LogisticRegression: 72.45/98.44/86.12/85.1
            # 4 - LinearDiscriminantAnalysis: 69.86
            # 5 - DecisionTree: 40.79
            # 6 - GaussianNB: 61.8
            # 7 - GaussianProcess: 66.82 (очень долго)
            # 8 - SVC: 75.65/97.99/87.08
            # 9 - QuadraticDiscriminantAnalysis: 10.35
            # 10 - MLP: 66.97
            # 11 - KNeighbors(12): 70.78/97.54/83.01
            # 12 - SVC (kernel = 'linear'): 75.49/97.99/86.72
            # 13 - PassiveAggressive: 69.86/99.11/86.60
            # 14 - Perceptron: 61.34/97.32/84.33
            # 15 - MultinomialNB: 67.88/98.44/83.37
            sklearn_cls = 0
    if vec_model in [100, 101]:
        pre = 'word2vec'
        if vec_model == 100:
            fn_wv = path + pre_ds + 'w2v.model'
        else:
            size = 100
            fn_wv = 'word2vec/b_w2v_txt.txt'
            if data_set != 2:
                print('Плохой набор данных')
                exit()
    elif vec_model == 200: # doc2vec
        pre = 'doc2vec'
        fn_wv = path + pre_ds + 'd2v.model'
    elif vec_model == 20: # fasttext
        pre = 'fasttext'
        fn_wv = path + pre_ds + 'ft.model'
        size = min(256, size)
    elif vec_model == 30: # fasttext2
        pre = 'fasttext_2'
        fn_wv = path + pre_ds + 'ft2.bin'
        if step == 3: step = 5
    elif vec_model == 40: # GloVe
        pre = 'glove'
##        size = 200
##        fn_wv = 'glove/glove_6B_' + str(size) + 'd.txt'
        size = 300
        fn_wv = 'glove/b_glove.txt'
        if data_set != 2:
            print('Плохой набор данных')
            exit()
    elif vec_model == 50: # LDA
        pre = 'LDA'
        lda_big = True
        n_components = 768 if lda_big else num_classes
        print('n_components =', n_components)
        fn_wv = path + pre_ds + pre + '.bin'
    elif vec_model == 60: # Части слов
        pre = 'КЧС'
        # Число признаков в модели слова
        n_attrs = 4 # n_attrs = 4: слово, 2, 3 и 4 последние буквы
        use_ends = True
        if not use_ends: sum_mix = 2
    elif vec_model == 7: # One-hot
        pre = 'One-hot'
        if data_set == 5: sum_mix = 0
        if sum_mix == 2: nw_in_doc_max = 15
    elif vec_model == 8 or vec_model == 9: # CountVectorizer, TfidfVectorizer
        pre = 'CountVectorizer' if vec_model == 8 else 'TfidfVectorizer'
        pipe = not True # Флаг применения Pipeline
        binary = not True
        if use_nn: pipe = False
    elif vec_model == 10: # Случайные векторы размера size
        pre = 'CB'
        # Замена слов на сумму слов-векторов с накоплением
        acc = not True # accumulation
        if acc:
            sum_mix = 0
            split_first = True
```

```
    elif vec_model == 11:
        pre = 'BERT_CLS'
        fn_b_vecs = 'b_vecs_cls'
    elif vec_model == 12:
        pre = 'BERT_1'
        fn_b_vecs = path + 'b_vecs'
    elif vec_model == 13:
        pre = 'BERT_4'
        fn_b_vecs = 'b_vecs4'
    elif vec_model == 14:
        pre = 'BERT_SEP'
        fn_b_vecs = 'b_vecs_sep'
    elif vec_model == 15: # Частотно-классовая
        pre = 'ЧКМ'
        split_first = True
        if sum_mix == 2: sum_mix = 0
    if vec_model in [11, 12, 13, 14]: # BERT
        if split_first:
            fn_b_vecs_t = path + pre_ds + fn_b_vecs + '_t' + '.txt'
            fn_b_vecs_v = path + pre_ds + fn_b_vecs + '_v' + '.txt'
        else:
            fn_b_vecs = path + pre_ds + fn_b_vecs + '.txt'
    if step == 4 and vec_model not in [8, 9]: pipe = False # Bag of words
    print('Модель', pre)
elif step == 5:
    fn_wv = path + pre_ds + 'ft2.bin'
    # 1 - создание обучающего и проверочного множеств; создание и сохранение модели
    # 2 - загрузка и проверка модели
    stp_ft2 = 1
fn_c = 'corp0.txt' # Тексты. Перед текстом имя класса, например, '#культура '
fn_c_short = 'corp_short.txt' # Сокращенный корпус
fn_x = path + pre_ds + 'x.txt'
fn_y = path + pre_ds + 'y.txt'
fn_d = path + pre_ds + 'dict.txt'
fn_x_lab = path + pre_ds + 'x_lab.txt'
fn_d_a = path + pre_ds + 'dict_a.txt' # Словарь кодов слова и его частей (атрибутов)
fn_in_cls = path + pre_ds + 'in_cls.txt' # Размеры классов
fn_xt = path + pre_ds + 'x_t.txt'
fn_yt = path + pre_ds + 'y_t.txt'
fn_xv = path + pre_ds + 'x_v.txt'
fn_yv = path + pre_ds + 'y_v.txt'
if data_set == 0:
    dict_cls = {'автомобили':0, 'здоровье':1, 'культура':2, 'наука':3,
            'недвижимость':4, 'политика':5, 'происшествия':6, 'реклама':7,
            'семья':8, 'спорт':9, 'страна':10, 'техника':11, 'экономика':12}
elif data_set == 1:
# ('earn',3582),('acq',2420),('money-fx',682),('crude',539),('grain',536),
# ('trade',472),('interest',339),('ship',209),('money-supply',177),('sugar',154),
# ('gnp',127),('coffee',126),('gold',123),('veg-oil',94),('cpi',86),
# ('oilseed',80),('cocoa',67),('copper',62),('reserves',62),('bop',60),
# ('livestock',58),('ipi',57),('jobs',57),('alum',53),('iron-steel',52),
# ('nat-gas',51),('dlr',46),('rubber',41),('gas',38),('tin',32),('carcass',29),
# ('pet-chem',29),('cotton',28),('wpi',27),('retail',23),('wheat',22),
# ('meal-feed',22),('orange',22),('zinc',21),('housing',19),
# ('strategic-metal',19),('lead',19),('hog',17),('heat',16),('lei',16),('silver',16)
    if step == 6:
        dict_cls = {'cocoa':0, 'grain':1, 'veg-oil':2, 'earn':3, 'acq':4, 'wheat':5,
                'copper':6, 'housing':7, 'money-supply':8, 'coffee':9, 'sugar':10,
                'trade':11, 'reserves':12, 'ship':13, 'cotton':14, 'carcass':15, 'crude':16,
                'nat-gas':17, 'cpi':18, 'money-fx':19, 'interest':20, 'gnp':21, 'meal-feed':22,
                'alum':23, 'oilseed':24, 'gold':25, 'tin':26, 'strategic-metal':27, 'livestock':28,
                'retail':29, 'ipi':30, 'iron-steel':31, 'rubber':32, 'heat':33, 'jobs':34, 'lei':35,
                'bop':36, 'zinc':37, 'orange':38, 'pet-chem':39, 'dlr':40, 'gas':41, 'silver':42,
                'wpi':43, 'hog':44, 'lead':45}
    else:
        # [[1, 0], [3, 1], [4, 2], [8, 3], [9, 4], [10, 5], [11, 6], [13, 7],
        # [16, 8], [19, 9], [20, 10], [21, 11], [25, 12]] - переименования
        # ('earn',3582),('acq',2420),('money-fx',682),('crude',539),('grain',536),
        # ('trade',472),('interest',339),('ship',209),('money-supply',177),('sugar',154),
        # ('gnp',127),('coffee',126),('gold',123)
        dict_cls = {'grain':0, 'earn':1, 'acq':2, 'money-supply':3, 'coffee':4,
                'sugar':5, 'trade':6, 'ship':7, 'crude':8, 'money-fx':9,
                'interest':10, 'gnp':11, 'gold':12}
    fn_x_w = path + pre_ds + 'x_w.txt' # fn_x_w - файл со словами, а не с кодами
    if step == 4:
        if vec_model == 60 or vec_model == 20: fn_x = fn_x_w # data_set == 1
elif data_set == 2:
    dict_cls = {'business':0, 'entertainment':1, 'politics':2, 'sport':3, 'tech':4}
elif data_set == 3:
    dict_cls = {'business':0, 'entertainment':1, 'politics':2, 'sport':3, 'tech':4,
            'food':5, 'graphics':6, 'historical':7, 'medical':8, 'space':9}
elif data_set == 4:
```

```
    dict_cls = {'contracts':0, 'dividend':1, 'earnings':2, 'joint_venture':3,
            'labor_union':4, 'lawsuit':5, 'management_changes':6, 'product':7}
elif data_set == 5:
    dict_cls = {'neg':0, 'pos':1}
num_classes = len(dict_cls) # Число классов
#
# Вычисление точности по вектору частоты
# (вектору вероятности принадлежности документа классу)
def one_eval(knd, x, y, sum_mix = 0, nv_in_doc_mix = 10, cls_numb = None):
    simple = True
    if not simple:
        from scipy.spatial import distance
        lst_cls = []
        for cls in range(num_classes):
            v = np.zeros(num_classes)
            v[cls] = 1
            lst_cls.append(v)
    n_true = 0
    for vec, cls in zip(x, y):
        if sum_mix == 1:
            v = np.zeros(num_classes)
            nvi, nva = 0, num_classes
            for nv in range(nv_in_doc_mix):
                v += np.array(vec[nvi:nva])
                nvi += num_classes
                nva += num_classes
            vec = v / nv_in_doc_mix
        if cls_numb is not None: cls = cls_numb[cls] # LDA
        if simple:
            if cls == vec.argmax(): n_true += 1
        else:
            lst_dst = [distance.cosine(vec_c, vec) for vec_c in lst_cls]
            if cls == np.array(lst_dst).argmin(): n_true += 1
    print('Точность на ' + knd + ' множестве', round(n_true / len(y) * 100, 2))
# Усреднение и конкатенация
def add_in_mix(c, lst_v, lst_nw, nv):
    v = lst_v[nv] + c
    lst_v[nv] = v
    lst_nw[nv] += 1
    nv += 1
    if nv == nv_in_doc_mix: nv = 0
    return lst_v, lst_nw, nv
def start_with_mix(len_mix, nv_in_doc_mix):
    lst_v = [np.zeros(len_mix)] * nv_in_doc_mix
    lst_nw = [0] * nv_in_doc_mix
    return lst_v, lst_nw
def do_avg_mix(lst_v, lst_nw, x_trn_vl_c):
    lst_v = [v / nw for v, nw in zip(lst_v, lst_nw)]
    lst_d = []
    for v in lst_v: lst_d.extend(v)
    x_trn_vl_c.append(lst_d)
# Загрузка текстового файла в список
def read_txt_f(fn, encoding = 'utf-8', no_n = True, say = True, to_int = False):
    with open(fn, 'r', encoding = encoding) as f:
        lst = f.readlines() # <class 'list'>
    if no_n:
        lst = [x.replace('\n', '') for x in lst]
    if to_int:
        lst = [int(x) for x in lst] # При чтении меток
    if say: print('Прочитан файл', fn, 'с числом строк', len(lst))
    return lst
# Сохранение списка в текстовый файл
def add_to_txt_f(lst, fn):
    print('Создан файл', fn, 'с числом строк', len(lst))
    with open(fn, 'w', encoding = 'utf-8') as f:
        for val in lst: f.write((val + '\n') if val.find('\n') == -1 else val)
# Формирование из списка слов словаря с элементами {слово:код слова}
def make_dict_from_list(lst, k0):
    dict_x = {}
    k = k0
    for x in lst:
        k += 1
        if dict_x.get(x) is None:
            dict_x.update({x : k})
    return dict_x
# Обработка строки русского текста
def preprocess_s(s):
    s = s.lower()
    s = s.replace('ё', 'е')
    # Оставляем только русские строчные буквы, остальное заменяем пробелами
    s = re.sub('[^а-я]', ' ', s)
    # Заменяем одиночные буквы на пробелы
```

```
    s = re.sub(r'\b[а-я]\b', ' ', s)
    s = re.sub(' +', ' ', s) # Заменяем несколько пробелов одним
    return s.strip() # Удаляем начальный и конечные пробелы
# Обработка строки английского текста
def preprocess_en(s):
    s = s.lower()
    # Оставляем только английские строчные буквы, остальное заменяем пробелами
    s = re.sub('[^a-z]', ' ', s)
    # Заменяем одиночные буквы на пробелы
    s = re.sub(r'\b[a-z]\b', ' ', s)
    s = re.sub(' +', ' ', s) # Заменяем несколько пробелов одним
    return s.strip() # Удаляем начальный и конечные пробелы
def load_save_data():
    import sqlite3
    dict_cls = {'auto':0, 'culture':0, 'economics':0, 'health':0,
                'incident':0, 'politics':0, 'realty':0, 'reclama':0,
                'science':0, 'social':0, 'sport':0, 'tech':0, 'woman':0}
    dict_cls2 = {'auto':'автомобили', 'culture':'культура', 'economics':'экономика',
                 'health':'здоровье', 'incident':'происшествия',
                 'politics':'политика', 'realty':'недвижимость',
                 'reclama':'реклама', 'science':'наука', 'social':'страна',
                 'sport':'спорт', 'tech':'техника', 'woman':'семья'}
    e = '\n'
    r = '\r'
    b = ' '
    conn = sqlite3.connect(db_name) # sqlite3.Connection
    c = conn.cursor() # sqlite3.Cursor
    res = c.execute('select * from data') # sqlite3.Cursor
##    print(res.description) # 'id', 'txt', 'tag'
    ft = open(fn_c, 'w', encoding = 'utf-8') # Результирующий файл
    n = l_max = n_words = 0
    for row in res: # type(row) - tuple
        n += 1
        txt = row[1]
        cls = row[2]
        dict_cls[cls] += 1
        txt = txt.replace(e, b)
        txt = txt.replace(r, '. ')
        txt = txt.replace(' .', '.')
        txt = re.sub('\.+', '.', txt)
        txt = re.sub(' +', b, txt) # Заменяем несколько пробелов одним
        n_words += (txt.count(b) + 1)
        l_max = max(l_max, len(txt))
        ft.write('#' + dict_cls2[cls] + ' ' + txt.strip() + e)
    conn.close()
    ft.close()
    print('Всего текстов:', n) # 3196
    print('Максимальная длина экземпляра текста:', l_max) # 30561
    print('Среднее число слов в экземпляре текста:', int(n_words / n)) # 239
    for itm in dict_cls.items():
        print(itm[0], dict_cls2[itm[0]], itm[1])
print('step =', step, '/ data_set =', data_set)
if step == -1:
    load_save_data()
elif step == 1: # Сокращаем корпус
    print('Доступ запрещен')
    exit()
    ft = open(fn_c, 'r', encoding = 'utf-8')
    x_trn = []
    ns = nb = 0
    while True:
        s = ft.readline()
        if not s: break
        s = preprocess_s(s)
        lst_s = s.split()
        len_s = len(lst_s)
        if len_s < nw_in_doc_min:
            ns += 1
            continue
        if len_s > nw_in_doc_max:
            nb += 1
            lst_s = lst_s[:nw_in_doc_max + 1]
        s2 = ''
        for w in lst_s:
            s2 += (w + ' ')
        x_trn.append('#' + s2.rstrip())
    ft.close()
    add_to_txt_f(x_trn, fn_c_short)
    print('Число коротких и длинных строк', ns, 'и', nb)
elif step == 2:
    # Заменяет слова леммами
    def to_normal_form(morph, s):
```

```
    s2 = s.split() # Список слов строки s
    s = ''
    for w in s2:
        w = morph.parse(w)[0].normal_form
        s += (' ' + w)
    return s.lstrip()
# Пополнение словарей dict_txt и dict_in_cls
def add_in_dicts(cls, dict_in_cls, nw_in_doc_max, doc, dict_txt):
    in_cls = dict_in_cls.get(cls)
    if in_cls is None:
        dict_in_cls[cls] = 1
    else:
        dict_in_cls[cls] += 1
    lst_t = doc.split()
    if nw_in_doc_max > 0:
        if len(lst_t) > nw_in_doc_max:
            lst_t = lst_t[:nw_in_doc_max]
    for w in lst_t: dict_txt[w] = 1 # Словарь корпуса
    doc = ''
    for w in lst_t: doc += (w + ' ')
    doc = doc.rstrip()
    return doc
t0 = time.time()
print('Подготовка данных')
dict_txt = {} # Словарь корпуса
dict_in_cls = {} # Словарь классов: {номер класса, число документов в классе}
x_trn, x_trn_lab, y_trn = [], [], [] # Списки для документов и меток корпуса
if data_set == 0: # ru
    if lem:
        import pymorphy2
        morph = pymorphy2.MorphAnalyzer()
    if lem: print('Лемматизация')
    ft = open(fn_c_short, 'r', encoding = 'utf-8')
    while True:
        doc = ft.readline()
        if not doc: break
        if len(doc.strip()) == 0: continue
        p = doc.find(' ')
        cls = doc[1:p]
        doc = doc[p + 1:]
        doc = preprocess_s(doc)
        if lem: doc = to_normal_form(morph, doc)
        doc = add_in_dicts(cls, dict_in_cls, nw_in_doc_max, doc, dict_txt)
        cls = str(dict_cls[cls])
        y_trn.append(cls)
        x_trn.append(doc)
        x_trn_lab.append('__label__' + cls + ' ' + doc)
    ft.close()
elif data_set == 1: # reuters
    x = read_txt_f(fn_x)
    y = read_txt_f(fn_y)
    lst_r_dict = read_txt_f('r_dict0.txt') # See reuters in Colab
    r_dict = {}
    for itm in lst_r_dict:
        itm = itm.split()
        r_dict[itm[0]] = itm[1]
    lst_in_cls = []
    for k, cls in zip(range(num_classes), dict_cls.keys()):
        lst_in_cls.append((cls, y.count(str(k))))
    lst_in_cls.sort(key = lambda t: t[1], reverse = True)
    for in_cls in lst_in_cls: print(in_cls[0], '-', in_cls[1])
    # vec_model in [20, 30, 6] части слов & fasttext & fasttext_2
    for d, cls in zip(x, y):
        d = d.split()
        doc = ''
        for w in d:
            w = r_dict.get(w)
            if w is not None:
                doc += (w + ' ')
        doc = doc.rstrip() # doc = preprocess_en(doc)
        doc = add_in_dicts(cls, dict_in_cls, nw_in_doc_max, doc, dict_txt)
        x_trn.append(doc)
        x_trn_lab.append('__label__' + cls + ' ' + doc)
    add_to_txt_f(x_trn, fn_x_w) # Все, кроме fasttext2
    add_to_txt_f(x_trn_lab, fn_x_lab) # fasttext_2
elif data_set in [2, 3, 4, 5]: # bbc, docs, e_docs, sentiments
    import os
    if data_set == 2:
        pth = 'bbc'
        i0 = 0
    elif data_set == 3:
        pth = 'docs'
```

```
            i0 = 0
        elif data_set == 4:
            pth = 'e_docs/data'
            i0 = 2
        elif data_set == 5:
            pth = 'sentiments/train'
            i0 = 0
        def one_data_set(pth, i0, x_trn_vl, x_trn_vl_lab, y_trn_vl):
            lst_dir = os.listdir(pth)
            lst_dir = [pth + '/' + dr for dr in lst_dir if dr.find('.') == -1]
            cls = -1
            for dr in lst_dir:
                cls += 1
                print('Формирование класса', cls)
                lst_fn = os.listdir(dr)
                lst_fn = [dr + '/' + fn for fn in lst_fn]
                for fn in lst_fn:
                    try:
                        lst_s = read_txt_f(fn, say = False)
                    except:
                        print('ERROR:', fn)
                        exit()
                    d = ''
                    for s in lst_s[i0:]:
                        if len(s.strip()) == 0: continue
                        s = preprocess_en(s)
                        d += s + ' '
                    d = d.split()
                    if len(d) < 10: continue
                    doc = ''
                    for w in d:
                        doc += w + ' '
                    doc = doc.rstrip()
                    s_cls = str(cls)
                    doc = add_in_dicts(s_cls, dict_in_cls, nw_in_doc_max, doc, dict_txt)
                    x_trn_vl.append(doc)
                    y_trn_vl.append(s_cls)
                    x_trn_vl_lab.append('__label__' + s_cls + ' ' + doc)
            return x_trn_vl, y_trn_vl, x_trn_vl_lab
        x_trn, y_trn, x_trn_lab = one_data_set(pth, i0, x_trn, x_trn_lab, y_trn)
        if data_set == 5:
            xv_all, xv_all_lab, yv_all = [], [], []
            xv_all, yv_all, xv_all_lab = one_data_set('sentiments/test', 0, xv_all, xv_all_lab, yv_all)
            xt_all = x_trn.copy()
            yt_all = y_trn.copy()
            x_trn.extend(xv_all)
            y_trn.extend(yv_all)
            x_trn_lab.extend(xv_all_lab)
    if data_set in [0, 2, 3, 4, 5]:
        add_to_txt_f(x_trn, fn_x) # Документы корпуса и их метки
        add_to_txt_f(y_trn, fn_y)
        add_to_txt_f(x_trn_lab, fn_x_lab) # fasttext2
        if data_set < 5:
            from sklearn.model_selection import train_test_split
            xt_all, yt_all, xv_all, yv_all = [], [], [], []
            # Делим каждый класс по отдельности, поскольку наборы данных несбалансированны
            for cls in range(num_classes):
                str_cls = str(cls)
                xt = [x for x, y in zip(x_trn, y_trn) if y == str_cls]
                yt = [str_cls] * len(xt)
                xt, xv, yt, yv = train_test_split(xt, yt, test_size = k_split, shuffle = False)
                xt_all.extend(xt)
                yt_all.extend(yt)
                xv_all.extend(xv)
                yv_all.extend(yv)
        add_to_txt_f(xt_all, fn_xt)
        add_to_txt_f(yt_all, fn_yt)
        add_to_txt_f(xv_all, fn_xv)
        add_to_txt_f(yv_all, fn_yv)
    lst_dict = list(dict_txt.keys())
    lst_dict.sort()
    add_to_txt_f(lst_dict, fn_d)
    lst_in_cls = [itm[0] + ' ' + str(itm[1]) for itm in dict_in_cls.items()]
    add_to_txt_f(lst_in_cls, fn_in_cls)
    print('Длительность подготовки данных:', round(time.time() - t0, 2))
elif step == 3: # word2vec, doc2vec, fasttext, LDA
    def make_doc_model(sg, size, window, min_cnt, n_iter, fn_c, fn_wv):
        # sg = 0 - используем модель CBOW (по умолчанию); sg = 1 - используем модель skip-gram
        # size - размерность признакового пространства
        # window - максимальное расстояние между текущим словом и словами около него
        # min_cnt - слово должно встречаться минимум min_cnt раз, чтобы модель его учитывала
        # n_iter - число итераций
```

```
        import multiprocessing
        if vec_model != 200:
            print('Создание ' + pre + '-модели по файлу', fn_c)
        workers = multiprocessing.cpu_count()
        print(' sg =', sg, '\n size =', size,
              '\n min_cnt =', min_cnt, '\n window =', window, '\n n_iter =', n_iter,
              '\n multiprocessing.cpu_count =', workers)
        start_time = time.time() # Время начала создания word2vec-модели
        if vec_model == 100:
            from gensim.models import Word2Vec
            from gensim.models.word2vec import LineSentence
            data = LineSentence(fn_c) # <class 'gensim.models.word2vec.LineSentence'>
            model = Word2Vec(data, size = size, window = window, min_count = min_cnt,
                             sg = sg, workers = workers, iter = n_iter) # iter = 5 (по умолчанию)
        elif vec_model == 200:
            from gensim.models.doc2vec import Doc2Vec, TaggedDocument
            from nltk.tokenize import word_tokenize as w_t
            print('Создание ' + pre + '-модели по файлам', fn_xt, 'и', fn_xv)
            x_trn = read_txt_f(fn_xt)
            x_vl = read_txt_f(fn_xv)
            print('Объединяем x_trn и x_vl')
            x_trn.extend(x_vl)
            tagged_corp = [TaggedDocument(words = w_t(d), tags = [str(i)])
                           for i, d in enumerate(x_trn)]
            model = Doc2Vec(vector_size = size,
                            window = window, min_count = min_cnt, epochs = n_iter)
            model.sg = 1 # Реализацция PV-DBOW
            # Создаем словарь корпуса
            model.build_vocab(tagged_corp)
            model.train(tagged_corp, total_examples = model.corpus_count,
                        epochs = model.epochs)
        elif vec_model == 20:
            from gensim.models import FastText
            model = FastText(corpus_file = fn_c, size = size, window = window,
                             min_count = min_cnt, sg = sg, workers = workers, iter = n_iter)
        print('Время создания', pre, 'модели:', round(time.time() - start_time, 0))
        print(pre + '-модель записана в файл', fn_wv)
        model.save(fn_wv)
        return model
    if data_set == 1 and vec_model == 20: fn_x = fn_x_w
    if vec_model == 50: # LDA
        from sklearn.decomposition import LatentDirichletAllocation
        from sklearn.feature_extraction.text import CountVectorizer
        if split_first:
            x_trn = read_txt_f(fn_xt)
            x_vl = read_txt_f(fn_xv)
            print('Объединяем x_trn и x_vl')
            x_trn.extend(x_vl)
        else:
            x_trn = read_txt_f(fn_x) # Корпус
        print('Создание LDA-векторов. vec_model =', vec_model)
        t0 = time.time()
        vec = CountVectorizer(token_pattern = '\w+', binary = False)
        x_trn = vec.fit_transform(x_trn)
        lda = LatentDirichletAllocation(n_components = n_components, random_state = 0)
        x_trn = lda.fit_transform(x_trn) # sklearn.decomposition._lda.LatentDirichletAllocation
        print('Время создания LDA-векторов:', round(time.time() - t0, 2))
        fn = open(fn_wv, 'wb')
        fn.write(np.float32(x_trn))
        fn.close()
    else:
        make_doc_model(sg, size, window, min_cnt, n_iter, fn_x, fn_wv)
elif step == 4: # Загрузка данных, формирование обучающего и оценочных множеств, классификация
    print('vec_model =', vec_model, '/ num_classes =', num_classes, '/ split_first =', split_first)
    print('sum_mix =', sum_mix, '/ nv_in_doc_mix =', nv_in_doc_mix, '/ use_nn =', use_nn)
    if split_first:
        x_trn = read_txt_f(path + fn_xt)
        y_trn = read_txt_f(path + fn_yt, to_int = True)
        x_vl = read_txt_f(path + fn_xv)
        y_vl = read_txt_f(path + fn_yv, to_int = True)
    else:
        from sklearn.model_selection import train_test_split
        if vec_model not in [11,12, 13, 14]: # BERT
            x_trn = read_txt_f(path + fn_x) # Корпус
        y_trn = read_txt_f(path + fn_y, to_int = True) # Метки документов корпуса
    if vec_model == 60: # Модель слова в виде кодов слова и его частей
        print('Модель слова в виде кодов слова и его частей. use_ends =', use_ends)
        # Пополняет словарь признаков по списку строковых признаков
        def update_dict_a(lst_a, dict_a, suff = '_'):
            lst_a = list(set(lst_a))
            for a in lst_a:
                a += suff
```

```
            if dict_a.get(a) is None:
                dict_a.update({a : 1})
        return dict_a
    # Формирует список числовых признаков по списку строковых признаков
    def make_lst_c(lst_a, dict_a, suff = '_'):
        lst_c = []
        for a in lst_a:
            a += suff
            c = dict_a.get(a)
            c = [0] if c is None else [c]
            lst_c.extend(c)
        return lst_c
    if use_ends:
        print('Число признаков в слове:', n_attrs)
        def add_ends(dict_a, n_attrs):
            lst_a = []
            for w in dict_a.keys():
                for n in range(2, n_attrs + 1):
                    lst_a.append(w[-n:])
            return update_dict_a(lst_a, dict_a)
        # Формируем список числовых признаков слова w
        def find_ends(w, dict_a, n_attrs):
            lst_a = []
            for n in range(2, n_attrs + 1):
                lst_a.append(w[-n:])
            return make_lst_c(lst_a, dict_a)
    else:
        def make_w_parts(w):
            lst_w_a = []
            lw = len(w)
            for i in range(lw):
                for k in range(1, lw - i):
                    lst_w_a.append(w[k:lw - i])
            return lst_w_a
        def add_parts(dict_a):
            lst_a = []
            for w in dict_a.keys():
                lst_a.extend(make_w_parts(w))
            return update_dict_a(lst_a, dict_a)
    # Список из слов словаря корпуса
    lst_dict = read_txt_f(fn_d)
    # Формируем начальный словарь атрибутов
    dict_a = make_dict_from_list(lst_dict, 0)
    # Добавляем в словарь части слов и получаем полный словарь атрибутов
    if use_ends:
        dict_a = add_ends(dict_a, n_attrs)
    else:
        dict_a = add_parts(dict_a)
    lst_dict_a = list(dict_a.keys())
    lst_dict_a.sort()
    add_to_txt_f(lst_dict_a, fn_d_a) # fn_d_a формируем для справки (не нужен)
    N = len(lst_dict_a) # Размер словаря атрибутов
    dict_a = make_dict_from_list(lst_dict_a, 0)
    # Приводим значения признаков к диапазону (0, 1] (нормализация)
    for a in dict_a.keys(): dict_a[a] /= N
    def mk_x_trn_vl_codes(x_trn_vl, n_attrs, dict_a, nw_in_doc_max):
        x_trn_vl_codes = []
        # Замена слов на числовые признаки слова и его частей
        nc_max = 0
        for doc in x_trn_vl:
            doc = doc.split()
            nw = 0
            if sum_mix == 0:
                lst_codes = np.zeros(n_attrs)
            else:
                lst_codes = []
            for w in doc:
                cw = dict_a.get(w)
                if cw is not None:
                    lst_c = [cw]
                    if use_ends:
                        lst_c.extend(find_ends(w, dict_a, n_attrs))
                    else:
                        lst_c.extend(make_lst_c(make_w_parts(w), dict_a))
                    nw += 1
                    if sum_mix == 0:
                        lst_codes += np.array(lst_c)
                    else:
                        lst_codes.extend(lst_c)
                    if nw == nw_in_doc_max: break
            if sum_mix == 0:
                lst_codes /= nw
```

```
            else:
                nc_max = max(nc_max, len(lst_codes))
            x_trn_vl_codes.append(lst_codes)
        if sum_mix > 0:
            # Пополняем нулями короткие списки числовых признаков
            for lst_codes in x_trn_vl_codes:
                dif = nc_max - len(lst_codes)
                if dif > 0:
                    lst_codes.extend([0 for i in range(dif)])
        return x_trn_vl_codes
    if split_first:
        x_trn = mk_x_trn_vl_codes(x_trn, n_attrs, dict_a, nw_in_doc_max)
        x_vl = mk_x_trn_vl_codes(x_vl, n_attrs, dict_a, nw_in_doc_max)
    else:
        x_trn = mk_x_trn_vl_codes(x_trn, n_attrs, dict_a, nw_in_doc_max)
elif vec_model == 7: # one-hot
    from sklearn.preprocessing import OneHotEncoder
    print('Модель one-hot')
    use_csr_matrix = False
    t0 = time.time()
    # Список из слов словаря корпуса
    lst_dict = read_txt_f(fn_d)
    # Сокращаем словарь в режиме проверки one-hot
    if use_csr_matrix:
        from scipy.sparse import csr_matrix
        l_mi, l_ma = 5, 7
        nw_in_doc_max = 10
    else:
        # Число букв в слове: минимальное и максимальное
        if sum_mix < 2:
            l_mi, l_ma = 2, 8
        else:
            l_mi, l_ma = 3, 7
    print('l_mi =', l_mi, 'l_ma =', l_ma)
    lst_dict = [w for w in lst_dict if len(w) > l_mi and len(w) < l_ma]
    len_dict = len(lst_dict)
    encoder = OneHotEncoder(sparse = False)
    arr_words = np.array(lst_dict).reshape(len_dict, 1)
    # class 'scipy.sparse.csr.csr_matrix' или class 'numpy.ndarray', если sparse = False
    arr_one_hot_words = encoder.fit_transform(arr_words)
    # Словарь корпуса: {слово:one-hot представление слова}
    dict_corp = {}
    for w, c in zip(lst_dict, arr_one_hot_words):
        dict_corp[w] = c
    arr_words = ''
    arr_one_hot_words = ''
    t2 = time.time()
    print('Длительность one-hot подготовки данных:', round(t2 - t0, 2))
    def make_trn_vl_1(x_trn_vl, y_trn_vl):
        x_trn_vl_one_hot = []
        y_trn_vl_one_hot = []
        # Замена слов на one-hot представления
        nc_max = 0
        for doc, cls in zip(x_trn_vl, y_trn_vl):
            doc = doc.split()
            nw = nv = 0
            if sum_mix == 0:
                cw_sum = np.zeros(len_dict)
            elif sum_mix == 1:
                lst_v, lst_nw = start_with_mix(len_dict, nv_in_doc_mix)
            else:
                lst_one_hot = []
            for w in doc:
                c = dict_corp.get(w)
                if c is not None:
                    nw += 1
                    if sum_mix == 0:
                        cw_sum += c
                    elif sum_mix == 1:
                        lst_v, lst_nw, nv = add_in_mix(c, lst_v, lst_nw, nv)
                    else:
                        lst_one_hot.extend(c)
                    if nw == nw_in_doc_max: break
            if nw > 0:
                if sum_mix == 0:
                    x_trn_vl_one_hot.append(cw_sum / nw)
                elif sum_mix == 1:
                    do_avg_mix(lst_v, lst_nw, x_trn_vl_one_hot)
                else:
                    nc_max = max(nc_max, len(lst_one_hot))
                    x_trn_vl_one_hot.append(lst_one_hot)
                y_trn_vl_one_hot.append(cls)
```

```
            if sum_mix == 2:
                # Пополняем нулями короткие one-hot списки
                for lst_one_hot in x_trn_vl_one_hot:
                    dif = nc_max - len(lst_one_hot)
                    if dif > 0:
                        lst_one_hot.extend([0 for i in range(dif)])
            if use_csr_matrix:
                x_trn_vl = []
                for lst_one_hot in x_trn_vl_one_hot:
                    x_trn_vl.append(csr_matrix(lst_one_hot))
                x_trn_vl_one_hot = x_trn_vl
            return x_trn_vl_one_hot, y_trn_vl_one_hot
        if split_first:
            x_trn, y_trn = make_trn_vl_1(x_trn, y_trn)
            x_vl, y_vl = make_trn_vl_1(x_vl, y_vl)
        else:
            x_trn = make_trn_vl_1(x_trn, y_trn)
        # Возможно, будет меньше проблем с памятью
        lst_dict = arr_one_hot_words = dict_corp = None
        print('Длительность замены слов на one-hot:', round(time.time() - t2, 2))
    elif vec_model in [8, 9]: # CountVectorizer, TfidfVectorizer
        if vec_model == 8:
            dm = 'CountVectorizer'
            from sklearn.feature_extraction.text import CountVectorizer
            vec = CountVectorizer(token_pattern = '\w+', binary = binary)
        elif vec_model == 9:
            dm = 'TfidfVectorizer'
            from sklearn.feature_extraction.text import TfidfVectorizer
            analyzer = 'word' # 'word' 'char'
            vec = TfidfVectorizer(analyzer = analyzer, binary = binary)
        print(dm + '. binary =', binary, '/ pipe =', pipe)
        if not pipe:
            if split_first:
                def make_shorter_trn_vl(x_trn_vl, l_mi, l_ma):
                    x = []
                    for d in x_trn_vl:
                        d2 = d.split()
                        d = ''
                        for w in d2:
                            if len(w) > l_mi and len(w) < l_ma: d += w + ' '
                        x.append(d.rstrip())
                    return x
                if data_set == 5:
                    l_mi, l_ma = 2, 8
                    print('l_mi =', l_mi, 'l_ma =', l_ma)
                    x_trn = make_shorter_trn_vl(x_trn, l_mi, l_ma)
                    x_vl = make_shorter_trn_vl(x_vl, l_mi, l_ma)
                len_trn = len(x_trn)
                print('Объединяем x_trn и x_vl')
                x_trn.extend(x_vl)
            # Векторизация данных (документов)
            # x_trn: class 'scipy.sparse.csr.csr_matrix'
            # x_trn[0]: class 'scipy.sparse.csr.csr_matrix'
            x_trn = vec.fit_transform(x_trn)
            if split_first:
                x_vl = x_trn[len_trn:]
                x_trn = x_trn[:len_trn]
                #print(x_trn.getnnz(), x_vl.getnnz())
    # word2vec; doc2vec; fasttext; fasttext_2, GloVe
    elif vec_model in [100, 101, 200, 20, 30, 40]:
        # Загрузка ранее созданной word2vec / fasttext / fasttext_2-модели
        if vec_model == 100:
            from gensim.models import Word2Vec
            model = Word2Vec.load(path + fn_wv)
        elif vec_model == 200:
            from gensim.models.doc2vec import Doc2Vec
            model = Doc2Vec.load(path + fn_wv)
        elif vec_model == 20:
            from gensim.models import FastText
            model = FastText.load(path + fn_wv) # FastText(vocab=51150, size=75, alpha=0.025)
        elif vec_model == 30: # fasttext_2
            import fasttext
            model = fasttext.load_model(fn_wv) # class 'fasttext.FastText._FastText'
            # list / include_freq = True: tupple
            all_words = model.get_words()
            wv = {}
            for w in all_words:
                wv[w] = model.get_word_vector(w)
        elif vec_model in [101, 40]: # word2vec, GloVe предобученные
            corp = read_txt_f(fn_x)
            dict_x = {} # Словарь векторов слов
            for d in corp:
```

```
            d = d.split()
            for w in d:
               dict_x[w] = 1
         t0 = time.time()
         wv = {}
         with open(fn_wv, 'r', encoding = 'utf-8') as f:
            for vec in f:
               vec = vec.split()
               word = vec[0] # Токен из файла fn_wv
               if dict_x.get(word) is not None:
                  vec = np.array(vec[1:], dtype = 'float32')
                  wv[word] = vec
         print('Размер словаря:', len(wv))
         print('Время создания словаря:', round(time.time() - t0, 2))
      if vec_model in [100, 20]:
         wv = model.wv # wv – индексированные векторы слов
         vocab = wv.vocab # Словарь модели
      tmp = [0 for i in range(size)] # Список из нулей для расширения коротких документов
      # Замена слов их векторами, или замена документа на сумму слов-векторов (sum_mix = 0),
      # или на вектор документа в случае doc2vec
      if vec_model == 200: # doc2vec
##          x_trn = [dv for dv in model.docvecs.doctag_syn0] # Устарело
         x_trn = [dv for dv in model.docvecs.vectors_docs]
         if split_first:
            len_trn = len(y_trn)
            x_vl = x_trn[len_trn:]
            x_trn = x_trn[:len_trn]
      else:
         def make_trn_vl_4(x_trn_vl, tmp):
            x_trn_vl_codes = [] # Список векторных представлений документов
            for doc in x_trn_vl:
               doc = doc.split()
               if sum_mix == 0:
                  cw_sum = np.zeros(size)
               elif sum_mix == 1:
                  lst_v, lst_nw = start_with_mix(size, nv_in_doc_mix)
               else:
                  lst_codes = [] # Список, содержащий векторное представление документа doc
               nv = nw = 0 # nw - число векторов, добавленных к cw_sum или в lst_codes
               for w in doc:
                  in_vocab = vocab.get(w) if vec_model in [100, 20] else wv.get(w)
                  if in_vocab is not None:
                     cw = wv[w] if vec_model in [100, 20] else in_vocab
                     if sum_mix == 0:
                        cw_sum += cw
                     elif sum_mix == 1:
                        lst_v, lst_nw, nv = add_in_mix(cw, lst_v, lst_nw, nv)
                     else:
                        lst_codes.extend(cw)
                     nw += 1
                     if nw == nw_in_doc_max: break
               if sum_mix == 0:
                  x_trn_vl_codes.append(cw_sum / nw)
               elif sum_mix == 1:
                  do_avg_mix(lst_v, lst_nw, x_trn_vl_codes)
               else:
                  # Дополняем короткие экземпляры до полной длины, равной nw_in_doc_max * size
                  for k in range(nw_in_doc_max - nw):
                     lst_codes.extend(tmp)
                  x_trn_vl_codes.append(lst_codes)
            return x_trn_vl_codes
         t0 = time.time()
         if split_first:
            x_trn = make_trn_vl_4(x_trn, tmp)
            x_vl = make_trn_vl_4(x_vl, tmp)
         else:
            x_trn = make_trn_vl_4(x_trn, tmp)
         print('Время подготовки данных:', round(time.time() - t0, 2))
   elif vec_model == 10:
      # Случайно генерируемые векторы слов
      print('Случайно генерируемые векторы слов')
      print('sum_mix =', sum_mix, '/ size =', size, '/ acc =', acc)
      t0 = time.time()
      tmp = [0 for i in range(size)] # Список из нулей для расширения коротких документов
      # Список из слов словаря корпуса
      lst_dict = read_txt_f(fn_d)
      dict_a = make_dict_from_list(lst_dict, 0) # Формируем словарь
      for w in dict_a.keys():
         dict_a[w] = list(np.random.uniform(-1, 1, [size]))
      def make_trn_vl_5(x_trn_vl, acc_tv):
         x_trn_vl_codes = [] # Список векторных представлений документов
         if acc_tv:
```

```
            bs = len(x_trn_vl) // 5
            nb = k = 0
        for doc in x_trn_vl:
            doc = doc.split()
            if acc_tv:
                k += 1
                if k > bs * nb:
                    cw_sum = np.zeros(size)
                    nb += 1
            else:
                if sum_mix == 0:
                    cw_sum = np.zeros(size)
                elif sum_mix == 1:
                    lst_v, lst_nw = start_with_mix(size, nv_in_doc_mix)
                else:
                    lst_codes = [] # Список, содержащий векторное представление документа doc
            nv = nw = 0 # nw - число векторов, добавленных к cw_sum или в lst_codes
            for w in doc:
                if dict_a.get(w) is not None:
                    cw = dict_a[w]
                    if sum_mix == 0:
                        cw_sum += cw
                    elif sum_mix == 1:
                        lst_v, lst_nw, nv = add_in_mix(cw, lst_v, lst_nw, nv)
                    else:
                        lst_codes.extend(cw)
                    nw += 1
                    if nw == nw_in_doc_max: break
            if sum_mix == 0:
                x_trn_vl_codes.append(cw_sum / nw)
            elif sum_mix == 1:
                do_avg_mix(lst_v, lst_nw, x_trn_vl_codes)
            else:
                # Дополняем короткие экземпляры до полной длины, равной nw_in_doc_max * size
                for k in range(nw_in_doc_max - nw):
                    lst_codes.extend(tmp)
                x_trn_vl_codes.append(lst_codes)
        return x_trn_vl_codes
    if split_first:
        x_trn = make_trn_vl_5(x_trn, acc)
        x_vl = make_trn_vl_5(x_vl, acc)
    else:
        x_trn = make_trn_vl_5(x_trn, acc)
    print('Время генерации:', round(time.time() - t0, 2))
elif vec_model in [11, 12, 13, 14]: # BERT
    print('11 - cls-векторы последнего блока; 12 - последний блок;')
    print('13 - 4 последних блока; 14 - sep-векторы последнего блока')
    def add_to_trn_vl(bert_vecs):
        x_trn_vl = []
        for vec in bert_vecs:
            vec = vec.split()
            x_trn_vl.append([float(v) for v in vec])
        return x_trn_vl
    if split_first:
        bert_vecs_t = read_txt_f(fn_b_vecs_t)
        bert_vecs_v = read_txt_f(fn_b_vecs_v)
        x_trn = add_to_trn_vl(bert_vecs_t)
        x_vl = add_to_trn_vl(bert_vecs_v)
    else:
        bert_vecs = read_txt_f(fn_b_vecs)
        x_trn = add_to_trn_vl(bert_vecs)
elif vec_model == 15: # Частотно-классовая модель текста
    dict_all_words = {}
    dict_cls_words = {}
    for cls in range(num_classes):
        dict_cls_words[cls] = {}
    for d, cls in zip(x_trn, y_trn):
        d = d.split()
        d_cls = dict_cls_words[cls]
        for w in d:
            if dict_all_words.get(w) is None:
                dict_all_words[w] = 1
            else:
                dict_all_words[w] += 1
            if d_cls.get(w) is None:
                d_cls[w] = 1
            else:
                d_cls[w] += 1
    dict_f_words = dict_all_words.copy()
    for w, k in dict_f_words.items():
        dict_f_words[w] = np.zeros(num_classes)
    for cls in range(num_classes):
```

```
            d_cls = dict_cls_words[cls]
            for w, f in d_cls.items():
                f = f / dict_all_words[w]
                dict_f_words[w][cls] = f
        # Формирование словаря нейтральных слов, то есть слов,
        # примерно одинаково часто встречаемых во всех классах
        dif_f = 0 # 0.3
        dict_n_words = {}
        if dif_f > 0:
            for w, arr_f in dict_f_words.items():
                avg = arr_f.sum() / num_classes
                arr_f /= avg
                lst_f = [f for f in arr_f if abs(1 - f) > dif_f]
                if len(lst_f)== 0:
                    dict_n_words[w] = 1
            print('Размер словаря нейтральных слов', len(dict_n_words))
        def make_trn_vl_15(lst, dict_f_words, sum_mix, nv_in_doc_mix):
            x_trn_vl = []
            for d in lst:
                d = d.split()
                nw = nv = 0
                if sum_mix == 0:
                    v = np.zeros(num_classes)
                elif sum_mix == 1:
                    lst_v, lst_nw = start_with_mix(num_classes, nv_in_doc_mix)
                for w in d:
                    if dif_f > 0 and dict_n_words.get(w) is not None: continue
                    f = dict_f_words.get(w)
                    if f is not None:
                        if sum_mix == 0:
                            v += f
                            nw += 1
                        elif sum_mix == 1:
                            lst_v, lst_nw, nv = add_in_mix(f, lst_v, lst_nw, nv)
                if sum_mix == 0:
                    x_trn_vl.append(v / nw)
                elif sum_mix == 1:
                    do_avg_mix(lst_v, lst_nw, x_trn_vl)
            return x_trn_vl
        if split_first:
            x_trn = make_trn_vl_15(x_trn, dict_f_words, sum_mix, nv_in_doc_mix)
            x_vl = make_trn_vl_15(x_vl, dict_f_words, sum_mix, nv_in_doc_mix)
            one_eval('проверочном', x_vl, y_vl, sum_mix, nv_in_doc_mix)
            one_eval('обучающем', x_trn, y_trn, sum_mix, nv_in_doc_mix)
        else:
            x_trn = make_trn_vl_15(x_trn, dict_f_words, sum_mix, nv_in_doc_mix)
            one_eval('всем', x_trn, y_trn, sum_mix, nv_in_doc_mix)
    elif vec_model == 50: # LDA
        with open(fn_wv, 'rb') as fn:
            x_trn = np.fromfile(fn , dtype = 'float32')
        len_trn = len(y_trn)
        # Меняем форму массива: делаем его двумерным
        x_trn.shape = (len_trn + len(y_vl), n_components) # Все векторы
        if n_components == num_classes:
            # Нужно LDA-темам найти соответствующие номера классов документов
            # Алгоритм:
            # Взять документы класса i
            # Найти точность классификаций для всех тем LDA и поставить классу i в соответствие
            # тему с наибольшей точностью классификации
            y = y_trn.copy()
            if split_first: y.extend(y_vl)
            lst_n = []
            for c in range(num_classes):
                lst = [0]*num_classes
                for cls, vec in zip(y, x_trn):
                    if c == cls:
                        c2 = vec.argmax()
                        lst[c2] += 1
                lst_n.append([c, lst])
            #for c in range(num_classes): print(lst_n[c])
            cls_numb = [0]*num_classes # Список соответствий класс - тема
            for c in range(num_classes):
                lst = lst_n[c][1]
                m = np.array(lst).argmax()
                cls_numb[c] = m
            if data_set == 0:
                cls_numb = [1, 11, 9, 4, 10, 7, 6 , 2 , 5 , 0, 3 , 8 , 12]
            elif data_set == 2:
                cls_numb = [2, 4, 1, 0, 3] # bbc
            elif data_set == 4:
                cls_numb = [7, 3, 5, 4, 1, 0, 6, 2]
            elif data_set == 5:
```

```
            cls_numb = [1, 0]
        def make_new_y(y_trn_vl, cls_numb):
            y = [cls_numb[cls] for cls in y_trn_vl]
            return y
        y_trn = make_new_y(y_trn, cls_numb)
    if split_first:
        x_vl = x_trn[len_trn:] # Делим векторы на проверочные и обучающие
        x_trn = x_trn[:len_trn]
        if n_components == num_classes:
            y_vl = make_new_y(y_vl, cls_numb)
            one_eval('проверочном', x_vl, y_vl) #, cls_numb = cls_numb)
            one_eval('обучающем', x_trn, y_trn) #, cls_numb = cls_numb)
    else:
        if n_components == num_classes:
            one_eval('всем', x_trn, y_trn, cls_numb = cls_numb)
if not split_first:
    x_trn, x_vl, y_trn, y_vl = train_test_split(x_trn, y_trn, test_size = k_split, shuffle = True)
if use_nn: # Классификатор - нейронная сеть
    print('use_conv =', use_conv)
    def nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc):
        from keras.models import Model
        from keras.layers import Input, Dropout, Flatten, Dense
        import keras.utils as ut
        len_trn = len(y_trn)
        len_vl = len(y_vl)
        if type(x_trn) == type([]):
            print('Преобразуем списки в массивы')
            x_trn = np.array(x_trn, dtype = 'float32')
            x_vl = np.array(x_vl, dtype = 'float32')
        # Переводим метки в one-hot представление
        y_trn = ut.to_categorical(y_trn, num_classes)
        y_vl = ut.to_categorical(y_vl, num_classes)
        if use_conv == 0 or use_conv == 5:
            inp_shape = (n_attrs_in_doc, )
        elif use_conv == 1 or use_conv == 3 or use_conv == 4:
            inp_shape = (n_attrs_in_doc, 1)
            x_trn = x_trn.reshape(len_trn, n_attrs_in_doc, 1)
            x_vl = x_vl.reshape(len_vl, n_attrs_in_doc, 1)
        elif use_conv == 2:
            if sum_mix == 2:
                sz_1 = nw_in_doc_max
                sz_2 = n_attrs_in_doc // sz_1
            elif sum_mix == 1:
                sz_1 = nv_in_doc_mix
                sz_2 = n_attrs_in_doc // s_1
            else:
                sz_1 = n_attrs_in_doc
                sz_2 = 1
            inp_shape = (sz_1, sz_2, 1)
            x_trn = x_trn.reshape(len_trn, sz_1, sz_2, 1)
            x_vl = x_vl.reshape(len_vl, sz_1, sz_2, 1)
        print('Формируем модель НС')
        inp = Input(shape = inp_shape, dtype = 'float32')
        x = Dropout(0.3)(inp)
        if use_conv == 1:
            from keras.layers import Conv1D, MaxPooling1D
            x = Conv1D(filters = 8, kernel_size = 3, padding = 'same', activation = 'relu')(x)
            x = MaxPooling1D(pool_size = 2, strides = 2, padding = 'same')(x)
        elif use_conv == 2:
            from keras.layers import Conv2D, MaxPooling2D
            x = Conv2D(filters = 8, kernel_size = 3, padding = 'same', activation = 'relu')(x)
            x = MaxPooling2D(pool_size = 2, strides = 2, padding = 'same')(x)
        elif use_conv == 3:
            if colab:
                from tensorflow.keras.layers import GRU
            else:
                from keras.layers import GRU
            x = GRU(32, return_sequences = True,
                    stateful = False, recurrent_initializer = 'glorot_uniform',
                    dropout = 0.3, recurrent_dropout = 0.3)(x)
        elif use_conv == 4:
            from keras.layers import LSTM
            x = LSTM(32, dropout = 0.3, recurrent_dropout = 0.3)(x)
        elif use_conv == 5:
            x = Dense(32, activation = 'relu')(x)
            x = Dropout(0.3)(x)
        if use_conv in [1, 2, 3]:
            x = Flatten()(x)
            x = Dropout(0.3)(x)
        x = Dense(32, activation = 'relu')(x)
        output = Dense(num_classes, activation = 'softmax')(x)
        model = Model(inp, output)
```

```
            model.summary()
            model.compile(optimizer = 'adam', loss = loss, metrics = ['accuracy'])
##          fn_nn = 'bbcd.hdf5'
##          model.save(fn_nn)
##          print('Модель классификатора записана в файл', fn_nn)
##          exit()
##          import keras.callbacks as cb
##          callbacks = [cb.EarlyStopping(monitor = 'val_acc', patience = 9)]
            start_time = time.time() # Время начала обучения
            # Обучаем НС
            model.fit(x_trn, y_trn, batch_size = 64, epochs = epochs, verbose = 2,
                    validation_data = (x_vl, y_vl)) # callbacks = callbacks
            print('Время обучения:', round(time.time() - start_time, 2))
            # Оценка модели НС на оценочных данных
            score = model.evaluate(x_vl, y_vl, verbose = 0)
            # Вывод потерь и точности
            print('Потери при тестировании: ', score[0])
            print('Точность при тестировании:', score[1])
        # epoch = 30
        if vec_model in [8, 9]: # CountVectorizer, TfidfVectorizer
            # До преобразования: x_trn, x_vl: class 'scipy.sparse.csr.csr_matrix'
            x_trn = np.float32(x_trn.toarray())
            x_vl = np.float32(x_vl.toarray())
        # После преобразования: x_trn, x_vl: class 'numpy.ndarray'
        n_attrs_in_doc = len(x_vl[0])
        print('Число признаков в документе:', n_attrs_in_doc)
        if vec_model not in [8, 9, 200]:
            if vec_model != 60: # 60 - ЧС
                if sum_mix == 0:
                    dvd = 1
                elif sum_mix == 1:
                    dvd = nv_in_doc_mix
                else:
                    dvd = nw_in_doc_max
                n_attrs = n_attrs_in_doc // dvd
            print('Максимальное число слов в документе:', nw_in_doc_max)
            print('Число признаков в слове:', n_attrs)
        nn_model_fit(x_trn, y_trn, x_vl, y_vl, num_classes, epochs, n_attrs_in_doc)
    else:
        print('sklearn_cls =', sklearn_cls)
        if sklearn_cls == 0:
            from sklearn.linear_model import SGDClassifier
            print('Функция потерь:', loss)
            doc_clf = SGDClassifier(loss = loss, max_iter = 1000, tol = 1e-3)
        elif sklearn_cls == 1:
            from sklearn.ensemble import AdaBoostClassifier
            doc_clf = AdaBoostClassifier()
        elif sklearn_cls == 2:
            from sklearn.ensemble import RandomForestClassifier
            doc_clf = RandomForestClassifier()
        elif sklearn_cls == 3:
            from sklearn.linear_model import LogisticRegression
            doc_clf = LogisticRegression(solver = 'lbfgs', # newton-cg
                                         max_iter = 500, multi_class = 'auto')
        elif sklearn_cls == 4:
            from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
            doc_clf = LinearDiscriminantAnalysis()
        elif sklearn_cls == 5:
            from sklearn.tree import DecisionTreeClassifier
            doc_clf = DecisionTreeClassifier()
        elif sklearn_cls == 6:
            from sklearn.naive_bayes import GaussianNB
            doc_clf = GaussianNB()
        elif sklearn_cls == 7:
            from sklearn.gaussian_process import GaussianProcessClassifier
            doc_clf = GaussianProcessClassifier()
        elif sklearn_cls == 8:
            from sklearn.svm import SVC
            doc_clf = SVC(gamma = 'scale', C = 1.0)
        elif sklearn_cls == 9:
            from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
            doc_clf = QuadraticDiscriminantAnalysis()
        elif sklearn_cls == 10:
            from sklearn.neural_network import MLPClassifier
            doc_clf = MLPClassifier(alpha = 0.01, max_iter = 200, solver = 'lbfgs', tol = 0.001)
        elif sklearn_cls == 11:
            from sklearn.neighbors import KNeighborsClassifier
            doc_clf = KNeighborsClassifier(12)
        elif sklearn_cls == 12:
            from sklearn.svm import SVC
            # C - штраф в случае ошибки
            doc_clf = SVC(kernel = 'linear', C = 0.025, probability = True)
```

```
        elif sklearn_cls == 13:
            from sklearn.linear_model import PassiveAggressiveClassifier
            doc_clf = PassiveAggressiveClassifier()
        elif sklearn_cls == 14:
            from sklearn.linear_model import Perceptron
            doc_clf = Perceptron()
        elif sklearn_cls == 15:
            from sklearn.naive_bayes import MultinomialNB
            doc_clf = MultinomialNB(alpha = 0.01)
        t0 = time.time()
        print('Обучение классификатора')
        if pipe:
            from sklearn.pipeline import Pipeline
            doc_clf = Pipeline([('vec', vec), ('clf', doc_clf)])
        doc_clf.fit(x_trn, y_trn) # Обучение классификатора
        print('Время обучения:', round(time.time() - t0, 2))
        print('Оценка точности классификации')
        if data_set < 5:
            import scipy
            for k in range(num_classes):
                x_v = [] # Список с данными класса k
                y_v = []
                for d, c in zip(x_vl, y_vl):
                    if c == k:
                        if type(d) in [type([]), type(np.ones(1)), type('a')]:
                            x_v.append(d)
                        else: # Преобразование scipy.sparse.csr.csr_matrix в массив
                            x_v.append(d.toarray()[0])
                        y_v.append(c)
                lv = len(y_v)
                if lv > 0:
                    score = doc_clf.score(x_v, y_v)
                else:
                    score = 0
                print('%s: %.2f. %s: %.4f' % ('Доля в классе', lv / (lv + y_trn.count(k)),
                                              list(dict_cls)[k].title(), score))
##          from sklearn.metrics import accuracy_score
##          predicted = doc_clf.predict(x_vl) # numpy.ndarray
##          print('Точность на проверочном множестве:', round(accuracy_score(y_vl, predicted), 4))
        # Оценка точности классификации
        score = doc_clf.score(x_vl, y_vl) # class 'numpy.float64'
        print('Точность на проверочном множестве:', round(score, 4))
        score = doc_clf.score(x_trn, y_trn) # class 'numpy.float64'
        print('Точность на обучающем множестве:', round(score, 4))
    elif step == 5: # fasttext_2
        # Text classification: val_acc = 0.2016
        word_representation_model = True
        print('Представление слов' if word_representation_model
              else 'Классификация документов')
        import fasttext
        if stp_ft2 == 1:
            print('Создание модели', pre)
            print(' sg =', sg, '\n size =', size,
                  '\n min_cnt =', min_cnt, '\n window =', window, '\n n_iter =', n_iter)
            t0 = time.time()
            if word_representation_model:
                # Обучение без учителя
                model = fasttext.train_unsupervised(fn_x, model = 'cbow', minCount = min_cnt,
                                                    dim = size, ws = window)
                # model - class 'fasttext.FastText._FastText'
            else:
                import random
                fn_trn = path + pre_ds + 'x_trn_lab.txt' # Обучающее множество
                fn_tst = path + pre_ds + 'x_tst_lab.txt' # Проверочное множество
                x_trn = read_txt_f(fn_x_lab)
                random.shuffle(x_trn) # Перемешиваем данные
                len_lst = len(x_trn)
                len_trn = int(1 - k_split * len_lst)
                len_tst = len_lst - len_trn
                lst_trn = x_trn[:len_trn]
                lst_tst = x_trn[len_trn:]
                add_to_txt_f(lst_trn, fn_trn) # Запись данных в файлы
                add_to_txt_f(lst_tst, fn_tst)
                hyper_params = {'lr':0.01, 'epoch':n_iter, 'wordNgrams':1,
                                'dim':size, 'ws':window}
                # Управляемое обучение (с учителем)
                model = fasttext.train_supervised(input = fn_trn, **hyper_params)
            model.save_model(fn_wv)
            print('Модель записана в файл', fn_wv)
            print('Время создания модели:', round(time.time() - t0, 0))
        else:
            model = fasttext.load_model(fn_wv) # class 'fasttext.FastText._FastText'
```

```
    if word_representation_model:
        all_words = model.get_words(include_freq = True)
        #print(all_words[0][1], all_words[1][1])
    else:
        acc = model.test(fn_trn)
        tst_acc = model.test(fn_tst)
        print('Точность на обучающем и проверочных множествах:',
            100 * round(acc[1], 4), 'и', 100 * round(tst_acc[1], 4))
##      tst_lb = model.test_label(fn_tst)
##      for itm in tst_lb.items(): print(itm)
        # Освобождаемся от меток перед обращением к predict
        x_tst_lab = read_txt_f(fn_tst)
        x_tst = []
        lst_n_cls = [0 for k in range(num_classes)]
        lst_n_cls_true = lst_n_cls.copy()
        for x in x_tst_lab:
            p = x.find(' ')
            lab = x[:p]
            cls = lab[-2:]
            if cls[0] == '_': cls = cls[1]
            cls = int(cls)
            x = x[p + 1:]
            p = model.predict(x, k = 1)
            lab_p = p[0][0]
            lst_n_cls[cls] += 1
            if lab == lab_p:
                lst_n_cls_true[cls] += 1
        n = sum(lst_n_cls)
        print('Точность прогноза (%) по классам на проверочном множестве размера', n)
        for n_cls, cls in zip(range(num_classes), dict_cls.keys()):
            in_cls = lst_n_cls[n_cls]
            acc_cls = 100 * round(lst_n_cls_true[n_cls] / in_cls, 4)
            print('{} - {} ({}): {}'.format(n_cls, cls, in_cls, acc_cls))
        print('Общая (%):', 100 * round(sum(lst_n_cls_true) / n, 4))
##      p = model.predict('генпрокуратура не дать', k = 2)
##      print(p)
##   txt = 'генпрокуратура не дать мельниченко пойти на заседание комиссия'
##   v = model.get_sentence_vector(txt); print(v, len(v))
##   w = 'дать'
##   v = model.get_sentence_vector(w); print(v, len(v))
##   l = model.get_line(txt); print(l)
##   s = model.get_subwords('бежит'); print(s)
elif step == 6: # Сокращаем reuters, оставляя классы с числом документов, более 100
    if data_set == 10:
        x_full = read_txt_f('r_x_full.txt')
        y_full = read_txt_f('r_y_full.txt', to_int = True)
        lst_in_cls = []
        for cls in range(num_classes):
            lst_in_cls.append(y_full.count(cls))
        x, y = [], []
        for d, cls in zip(x_full, y_full):
            if lst_in_cls[cls] > 100:
                x.append(d)
                y.append(cls)
        lst_in_cls = []
        for cls in range(num_classes):
            cnt = y.count(cls)
            if cnt > 0:
                lst_in_cls.append([cls, cnt])
        num_classes = len(lst_in_cls)
        lst_in_cls.sort()
        for cls in range(num_classes):
            lst_in_cls[cls][1] = cls
        for itm in lst_in_cls:
            cls_old = itm[0]
            cls_new = itm[1]
            for k in range(len(y)):
                if y[k] == cls_old: y[k] = -cls_new
        y = [-cls for cls in y]
        lst_in_cls = []
        for cls in range(num_classes):
            cnt = y.count(cls)
            if cnt > 0:
                lst_in_cls.append([cls, cnt])
        print(lst_in_cls)
        y = [str(cls) for cls in y]
        add_to_txt_f(x, fn_x)
        add_to_txt_f(y, fn_y)
    elif data_set == 1:
        x = read_txt_f(fn_x)
        y = read_txt_f(fn_y)
        yx = [[a, b] for a, b in zip(y, x)]
```

```
yx.sort()
x = [a[1] for a in yx]
y = [a[0] for a in yx]
add_to_txt_f(x, 'r_x2.txt')
add_to_txt_f(y, 'r_y2.txt')
```

Список литературы

1. Башмаков А. И., Башмаков И. А. Интеллектуальные информационные технологии. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2005. – 304 с.
2. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. – СПб.: Питер, 2018. – 480 с.
3. BBC Dataset. [Электронный ресурс] URL: http://mlg.ucd.ie/datasets/bbc.html (дата обращения: 01.01.2021).
4. Лемматизация. [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Лемматизация (дата обращения: 01.01.2021).
5. Бенгфорт Б., Билбро Р., Охеда Т. Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка. – СПб.: Питер, 2019. – 368 с.
6. TfidfVectorizer. [Электронный ресурс] URL: https://scikit-learn.org/stable/modules/generated /sklearn.feature_extraction.text.TfidfVectorizer.html (дата обращения: 01.01.2021).
7. Blei D. M., Ng A. Y., Jordan M. I. Latent Dirichlet Allocation. – 2003. [Электронный ресурс] URL: https://jmlr.org/papers /volume3/blei03a/blei03a.pdf (дата обращения: 01.01.2021).
8. Latent semantic analysis. [Электронный ресурс] URL: https://en.wikipedia.org /wiki/Latent_semantic_analysis#Latent_semantic_indexing (дата обращения: 01.01.2021).
9. Distributed Representations of Words and Phrases and their Compositionality / T. Mikolov et al. // arXiv, 2013. [Электронный ресурс] URL: http://arxiv.org/abs/1310.4546 (дата обращения: 01.01.2021).
10. Bengio Y., Schwenk H., Senecal JS et al. Neural probabilistic language models. In Innovations in Machine Learning, pp. 137–186. Springer, 2006.
11. Morin F., Bengio Y. Hierarchical probabilistic neural network language model. In Proceedings of the International Workshop on Artificial Intelligence and Statistics. – 2005, pp. 246–252.
12. Le Q., Mikolov T. Distributed Representations of Sentences and Documents. – 2014. [Электронный ресурс] URL: https://cs.stanford.edu/~quocle/paragraph_vector.pdf (дата обращения: 01.01.2021).
13. Bojanowski P. et al. Enriching Word Vectors with Subword Information. – 2017. [Электронный ресурс] URL: https://arxiv.org /pdf/1607.04606.pdf (дата обращения: 01.01.2021).
14. Pennington J., Socher R., Manning C. D. GloVe: Global Vectors for Word Representation. – 2014. [Электронный ресурс] URL: https://nlp.stanford.edu/pubs/glove.pdf (дата обращения: 01.01.2021).
15. Download pre-trained word vectors. [Электронный ресурс] URL: https://nlp.stanford.edu/projects/glove/ (дата обращения: 01.01.2021).
[16] Pretrained models. [Электронный ресурс] URL: https://huggingface.co/transformers/pretrained_models.html (дата обращения: 01.01.2021).
17. GLUE tasks. [Электронный ресурс] URL: https://gluebenchmark.com/tasks (дата обращения: 01.01.2021).
18. SquAD 2.0. [Электронный ресурс] URL: https://rajpurkar.github.io/SQuAD-explorer/ (дата обращения: 01.01.2021).
19. Lai G. et al. RACE: Large-scale Reading Comprehension Dataset From Examinations. – 2017. [Электронный ресурс] URL: https://arxiv.org/pdf/1704.04683.pdf (дата обращения: 01.01.2021).
20. Vaswani A. et al. Attention Is All You Need. – 2017. [Электронный ресурс] URL: https://arxiv.org/pdf/1706.03762.pdf (дата обращения: 01.01.2021).
21. Radford A. et al. Improving Language Understanding by Generative Pre-Training. – 2018. [Электронный ресурс] URL: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf (дата обращения: 01.01.2021).
22. Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. – 2019. [Электронный ресурс] URL: https://arxiv.org/pdf/1810.04805.pdf (дата обращения: 01.01.2021).
23. Добро пожаловать в Colaboratory. [Электронный ресурс] URL: https://colab.research.google.com/notebooks/intro.ipynb (дата обращения: 01.01.2021).
24. Hugging Face. Models. [Электронный ресурс] URL: https://huggingface.co/models?filter=ru&search=bert (дата обращения: 01.01.2021).
25. DeepPavlov. [Электронный ресурс] URL: https://deeppavlov.ai/ (дата обращения: 01.01.2021).
26. Martin L. et al. CamemBERT: a Tasty French Language Model. – 2019. [Электронный ресурс] URL: https://arxiv.org /pdf/1911.03894.pdf (дата обращения: 01.01.2021).
27. Le H. et al. FlauBERT: Unsupervised Language Model Pre-training for French. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/1912.05372.pdf (дата обращения: 01.01.2021).
28. Nguyen D. Q., Anh Tuan Nguyen A. T. PhoBERT: Pre-trained language models for Vietnamese. – 2020. [Электронный ресурс] URL: https://www.aclweb.org/anthology/2020.findings-emnlp.92.pdf (дата обращения: 01.01.2021).
29. Lan Z. et al. ALBERT: A lite BERT for self-supervised learning of language representations. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/1909.11942.pdf (дата обращения: 01.01.2021).
30. Liu Y. et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach. – 2019. [Электронный ресурс] URL: https://arxiv.org /pdf/1907.11692.pdf (дата обращения: 01.01.2021).
31. Dataset: cc_news. [Электронный ресурс] URL: https://huggingface.co/datasets/cc_news (дата обращения: 01.01.2021).
32. Rothe S., Narayan S., Severyn A. Leveraging Pre-trained Checkpoints for Sequence Generation Tasks. - 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/1907.12461.pdf (дата обращения: 01.01.2021).
33. Jiang Z. et al. ConvBERT: Improving BERT with Span-based Dynamic Convolution. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/2008.02496.pdf (дата обращения: 01.01.2021).
34. Lewis M. et al. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. – 2019. [Электронный ресурс] URL: https://arxiv.org/pdf/1910.13461.pdf (дата обращения: 01.01.2021).
35. Eddine M. K., Tixier A. J.-P., Vazirgiannis M. BARThez: a Skilled Pretrained French Sequence-to-Sequence Model. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/2010.12321.pdf (дата обращения: 01.01.2021).
36. He P. et al. DEBERTA: Decoding-enhanced BERT with disentangled attention. – 2021. [Электронный ресурс] URL: https://arxiv.org/pdf/2006.03654.pdf (дата обращения: 01.01.2021).
37. Sanh V. et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/1910.01108.pdf (дата обращения: 01.01.2021).
38. Clark K. et al. ELECTRA: Pre-training text encoders as discriminators rather than generators. – 2020. [Электронный ресурс] URL: https://openreview.net/pdf?id=r1xMH1BtvB (дата обращения: 01.01.2021).
39. Dai Z. et al. Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/2006.03236.pdf (дата обращения: 01.01.2021).
40. Beltagy I., Peters M. E., Cohan A. Longformer: The Long-Document Transformer. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/2004.05150.pdf (дата обращения: 01.01.2021).

41. Sun Z. et al. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/2004.02984.pdf (дата обращения: 01.01.2021).
42. Dai Z. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. – 2019. [Электронный ресурс] URL: https://arxiv.org/pdf/1901.02860.pdf (дата обращения: 01.01.2021).
43. Yang Z. et al. XLNet: Generalized Autoregressive Pretraining for Language Understanding. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/1906.08237.pdf (дата обращения: 01.01.2021).
44. Song K. et al. MPNet: Masked and Permuted Pre-training for Language Understanding. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/2004.09297.pdf (дата обращения: 01.01.2021).
45. Iandola F. N. et al. SqueezeBERT: What can computer vision teach NLP about efficient neural networks? – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/2006.11316.pdf (дата обращения: 01.01.2021).
46. Raffel C. et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/1910.10683.pdf (дата обращения: 01.01.2021).
47. Conneau A. et al. Unsupervised Cross-lingual Representation Learning at Scale. – 2020. [Электронный ресурс] URL: https://arxiv.org/pdf/1911.02116.pdf (дата обращения: 01.01.2021).
48. Набор данных для классификации документов. [Электронный ресурс] URL: https://www.kaggle.com/olegbartenyev/doc-cls (дата обращения: 01.01.2021).
49. Business articles classification dataset. [Электронный ресурс] URL: https://www.kaggle.com/arpytanshu/business-articles-classification-dataset (дата обращения: 01.01.2021).
50. Large Movie Review Dataset. [Электронный ресурс] URL: http://ai.stanford.edu/~amaas/data/sentiment/ (дата обращения: 01.01.2021).
51. Generalized Language Models. [Электронный ресурс] URL: https://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html (дата обращения: 01.01.2021).
52. NLPL word embeddings repository. [Электронный ресурс] URL: http://vectors.nlpl.eu/repository/ (дата обращения: 01.01.2021).

Список работ