Deep Learning School

Физтех-Школа Прикладной математики и информатики (ФПМИ) МФТИ

Задача определения частей речи, Part-Of-Speech Tagger (POS)

Мы будем решать задачу определения частей речи (POS-теггинга) с помощью скрытой марковской модели (HMM).

```
1 import nltk
2 import pandas as pd
3 import numpy as np
4 from collections import OrderedDict, deque
5 from nltk.corpus import brown
6 import matplotlib.pyplot as plt
```

Вам в помощь http://www.nltk.org/book/

Загрузим brown корпус

```
1 nltk.download('brown')
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
True
```

Существует множество наборов грамматических тегов, или тегсетов, например:

- НКРЯ
- Mystem
- UPenn
- OpenCorpora (его использует pymorphy2)
- Universal Dependencies

Существует не одна система тегирования, поэтому будьте внимательны, когда прогнозируете тег слов в тексте и вычисляете качество прогноза. Можете получить несправедливо низкое качество вашего решения.

На данный момент стандартом является **Universal Dependencies**. Подробнее про проект можно почитать вот тут, а про теги — вот тут

```
1 nltk.download('universal_tagset')
```

```
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
True
```

- ADJ: adjective
- ADP: adposition
- ADV: adverb
- AUX: auxiliary
- CCONJ: coordinating conjunction
- DET: determiner
- INTJ: interjection
- NOUN: noun
- NUM: numeral
- PART: particle
- PRON: pronoun
- PROPN: proper noun
- PUNCT: punctuation
- SCONJ: subordinating conjunction
- SYM: symbol
- VERB: verb
- X: other

Мы имеем массив предложений пар (слово-тег)

```
1 brown_tagged_sents = brown.tagged_sents(tagset="universal")
```

```
2 brown_tagged_sents
```

```
[[('The', 'DET'), ('Fulton', 'NOUN'), ('County', 'NOUN'), ('Grand', 'ADJ'), ('Jury',
```

Первое предложение

```
1 brown_tagged_sents[0]
```

```
[('The', 'DET'),
 ('Fulton', 'NOUN'),
 ('County', 'NOUN'),
 ('Grand', 'ADJ'),
 ('Jury', 'NOUN'),
 ('said', 'VERB'),
 ('Friday', 'NOUN'),
 ('an', 'DET'),
 ('investigation', 'NOUN'),
 ('of', 'ADP'),
 ("Atlanta's", 'NOUN'),
 ('recent', 'ADJ'),
 ('primary', 'NOUN'),
 ('election', 'NOUN'),
 ('produced', 'VERB'),
 ('``', '.'),
 ('no', 'DET'),
 ('evidence', 'NOUN'),
 ("''", '.'),
 ('that', 'ADP'),
 ('any', 'DET'),
 ('irregularities', 'NOUN'),
 ('took', 'VERB'),
 ('place', 'NOUN'),
 ('.', '.')]
```

Все пары (слово-тег)

```
1 brown_tagged_words = brown.tagged_words(tagset='universal')
2 brown_tagged_words
```

```
[('The', 'DET'), ('Fulton', 'NOUN'), ...]
```

Проанализируйте данные, с которыми Вы работаете. Используйте `nltk.FreqDist()` для подсчета частоты встречаемости тега и слова в нашем корпусе. Под частой элемента подразумевается кол-во этого элемента в корпусе.

```
1 # Приведем слова к нижнему регистру
2 brown_tagged_words = list(map(lambda x: (x[0].lower(), x[1]), brown_tagged_words))
```

```
1 print('Кол-во предложений: ', len(brown_tagged_sents))
2 tags = [tag for (word, tag) in brown_tagged_words] # наши теги
3 words = [word for (word, tag) in brown_tagged_words] # наши слова
```

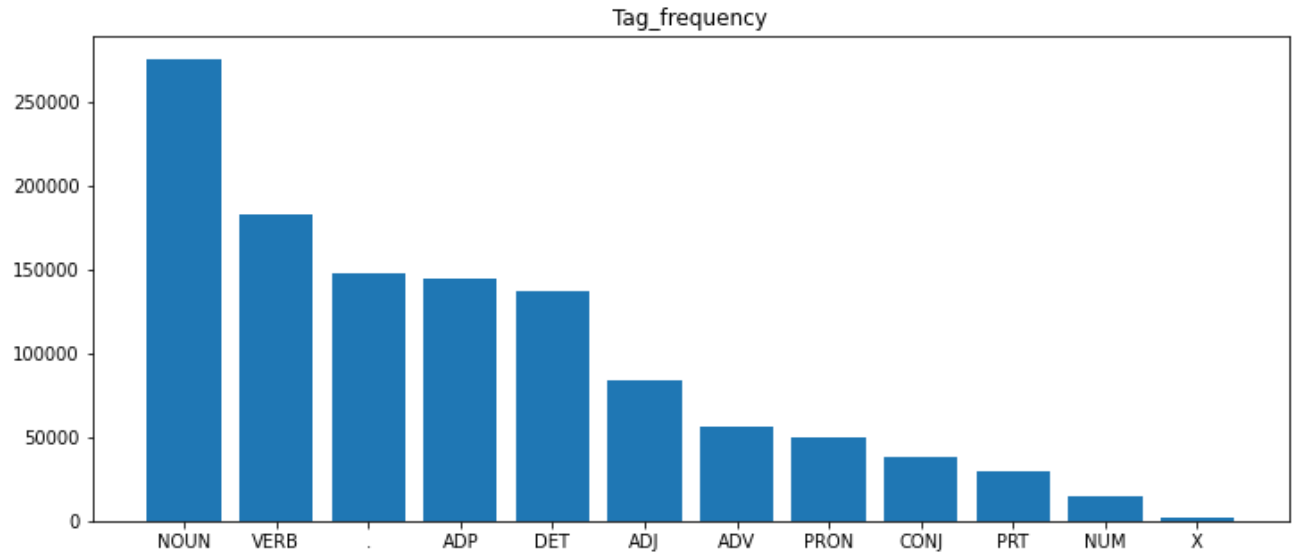
```
4
5 tag_num = pd.Series(nltk.FreqDist(tags)).sort_values(ascending=False) # тег - кол-во те
6 word_num = pd.Series(nltk.FreqDist(words)).sort_values(ascending=False) # слово - кол-в
```

```
Кол-во предложений:  57340
```

```
1 tag_num
```

```
NOUN    275558
VERB    182750
.       147565
ADP     144766
DET     137019
ADJ      83721
ADV      56239
PRON     49334
CONJ     38151
PRT      29829
NUM      14874
X         1386
dtype: int64
```

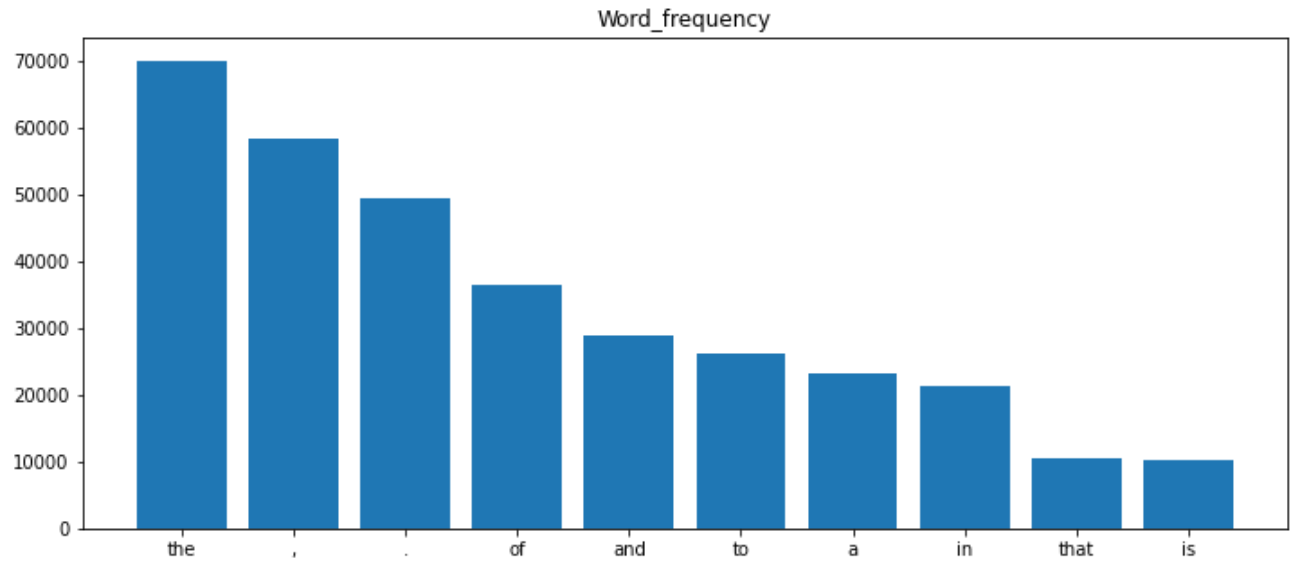
```
1 plt.figure(figsize=(12, 5))
2 plt.bar(tag_num.index, tag_num.values)
3 plt.title("Tag_frequency")
4 plt.show()
```




```
1 word_num[:5]
```

```
the    69971
,      58334
.      49346
of     36412
and    28853
dtype: int64
```

```
plt.figure(figsize=(12, 5))
plt.bar(word_num.index[:10], word_num.values[:10])
plt.title("Word_frequency")
plt.show()
```



Вопрос 1:

- Кол-во слова `cat` в корпусе?

```
word_num['cat']
```

```
23
```

Вопрос 2:

- Самое популярное слово с самым популярным тегом?
  (*сначала выбираете слова с самым популярным тегом, а затем выбираете самое популярное слово из уже выбранных*)

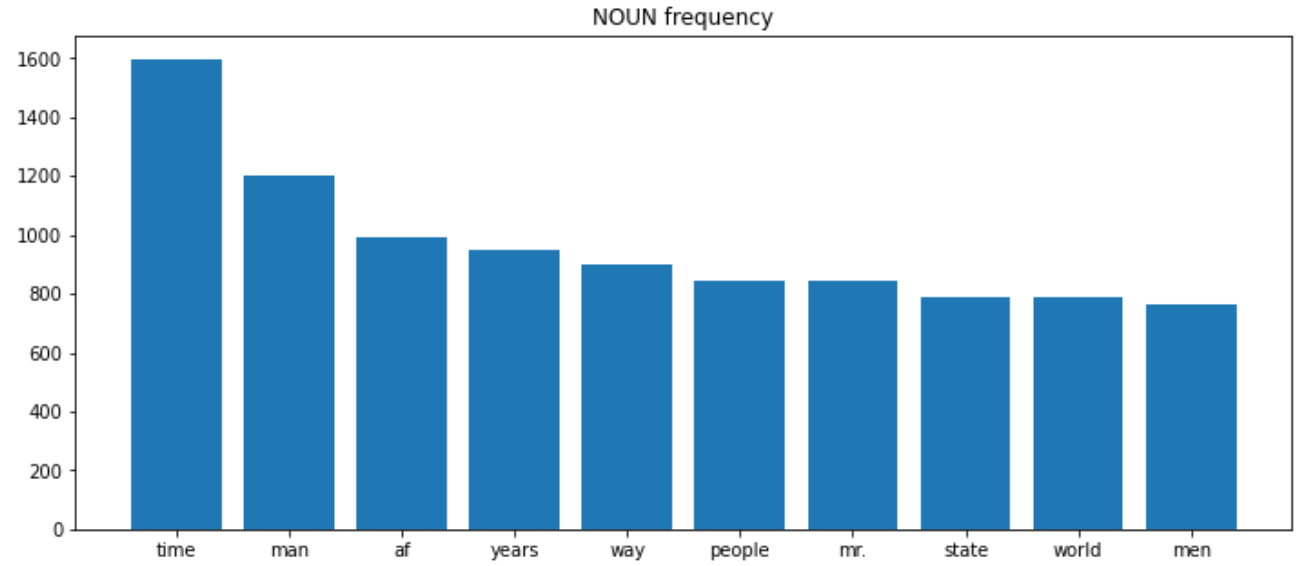
```
words_with_most_popular_tag = [word for (word, tag) in brown_tagged_words if tag == tag
words_with_most_popular_tag_num = pd.Series(nltk.FreqDist(words_with_most_popular_tag))
```

```
words_with_most_popular_tag_num.index[0]
```

```
'time'
```

```
plt.figure(figsize=(12, 5))
plt.bar(words_with_most_popular_tag_num.index[:10], words_with_most_popular_tag_num.val
plt.title(f"{tag_num.index[0]} frequency")
plt.show()
```

```
print()
print(f'Most popular {tag_num.index[0]}:', words_with_most_popular_tag_num.index[0])
```




```
Most popular NOUN: time
```

Впоследствии обучение моделей может занимать слишком много времени, работайте с подвыборкой, например, только текстами определенных категорий.

Категории нашего корпуса:

```
brown.categories()
```

```
['adventure',
 'belles_lettres',
 'editorial',
 'fiction',
 'government',
 'hobbies',
 'humor',
 'learned',
 'lore',
 'mystery',
 'news',
 'religion',
 'reviews',
 'romance',
 'science_fiction']
```

Будем работать с категорией humor

Сделайте случайное разбиение выборки на обучение и контроль в отношении 9:1.

```
# brown_tagged_sents = brown.tagged_sents(tagset="universal", categories='humor')
brown_tagged_sents = brown.tagged_sents(tagset="universal")

```

```
# Приведем слова к нижнему регистру
my_brown_tagged_sents = []
for sent in brown_tagged_sents:
    my_brown_tagged_sents.append(list(map(lambda x: (x[0].lower(), x[1]), sent)))
my_brown_tagged_sents = np.array(my_brown_tagged_sents)

from sklearn.model_selection import train_test_split
train_sents, test_sents = train_test_split(my_brown_tagged_sents, test_size=0.1, random
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: VisibleDeprecationWar
```

```
len(train_sents)
```

```
51606
```

```
len(test_sents)
```

```
5734
```

Метод максимального правдоподобия для обучения модели

- $S = s_0, s_1, \ldots, s_N$ - скрытые состояния, то есть различные теги
- $O = o_0, o_1, \ldots, o_M$ - различные слова
- $a_{i,j} = p(s_j|s_i)$ - вероятность того, что, находясь в скрытом состоянии s_i, мы попадем в состояние s_j (элемент матрицы A)
- $b_{k,j} = p(o_k|s_j)$ - вероятность того, что при скрытом состоянии s_j находится слово o_k(элемент матрицы B)

$$x_t \in O, y_t \in S$$

(x_t, y_t) - слово и тег, стоящие на месте $t \Rightarrow$

- X - последовательность слов
- Y - последовательность тегов

Требуется построить скрытую марковскую модель (class HiddenMarkovModel) и написать метод fit для настройки всех её параметров с помощью оценок максимального правдоподобия по размеченным данным (последовательности пар слово+тег):

- Вероятности переходов между скрытыми состояниями $p(y_t|y_{t-1})$ посчитайте на основе частот биграмм POS-тегов.
- Вероятности эмиссий наблюдаемых состояний $p(x_t|y_t)$ посчитайте на основе частот "POS-тег - слово".
- Распределение вероятностей начальных состояний $p(y_0)$ задайте равномерным.

Пример $X = [x_0, x_1], Y = [y_0, y_1]$:

$$p(X,Y) = p(x_0, x_1, y_0, y_1) = p(y_0) \cdot p(x_0, x_1, y_1 | y_0) = p(y_0) \cdot p(x_0 | y_0) \cdot p(x_1, y_1 | x_0, y$$
$$= p(y_0) \cdot p(x_0 | y_0) \cdot p(y_1 | x_0, y_0) \cdot p(x_1 | x_0, y_0, y_1) = (\text{в силу условий нашей модели}) =$$
$$= p(y_0) \cdot p(x_0 | y_0) \cdot p(y_1 | y_0) \cdot p(x_1 | y_1) \Rightarrow$$

Для последовательности длины $n+1$:

$$p(X,Y) = p(x_0 \dots x_{n-1}, y_0 \dots y_{n-1}) \cdot p(y_n | y_{n-1}) \cdot p(x_n | y_n)$$

Алгоритм Витерби для применения модели

Требуется написать метод .predict для определения частей речи на тестовой выборке. Чтобы использовать обученную модель на новых данных, необходимо реализовать алгоритм Витерби. Это алгоритм динамиеского программирования, с помощью которого мы будем находить наиболее вероятную последовательность скрытых состояний модели для фиксированной последовательности слов:

$$\hat{Y} = \arg\max_Y p(Y|X) = \arg\max_Y p(Y, X)$$

Пусть $Q_{t,s}$ - самая вероятная последовательность скрытых состояний длины t с окончанием в состоянии s. $q_{t,s}$ - вероятность этой последовательности.

$$(1)\ q_{t,s} = \max_{s'} q_{t-1,s'} \cdot p(s|s') \cdot p(o_t|s)$$

$Q_{t,s}$ можно восстановить по argmax-ам.

```
class HiddenMarkovModel:
    def __init__(self):

        pass

    def fit(self, train_tokens_tags_list):
        """
        train_tokens_tags_list: массив предложений пар слово-тег (выборка для train)
        """
        tags = [tag for sent in train_tokens_tags_list
                for (word, tag) in sent]
        words = [word for sent in train_tokens_tags_list for (word, tag) in sent]

        tagged_words = [(word, tag) for sent in train_tokens_tags_list
                                    for (word, tag) in sent]

        tag_num = pd.Series(nltk.FreqDist(tags)).sort_values(ascending=False)
        word_num = pd.Series(nltk.FreqDist(words)).sort_values(ascending=False)

        self.tags = tag_num.index
        self.words = word_num.index

        mpv = self.words[0] # most popular word
        mpt = self.tags[0]  # most popular tag
        # words with most popular tag
        words_with_most_popular_tag = [word for (word, tag) in tagged_words if tag == m
```

```
27          # most popular word
28          mpwmpt = pd.Series(nltk.FreqDist(words_with_most_popular_tag)).sort_values(asce
29
30
31          self.most_popular_tag = mpt
32          self.most_popular_word = mpv
33          self.most_popular_word_from_most_popular_tag = mpwmpt
34
35
36
37
38
39          A = pd.DataFrame({'{}'.format(tag) : [0] * len(tag_num) for tag in tag_num.inde
40          B = pd.DataFrame({'{}'.format(tag) : [0] * len(word_num) for tag in tag_num.ind
41
42          # Вычисляем матрицу A и B по частотам слов и тегов
43
44          # sent - предложение
45          # sent[i][0] - i слово в этом предложении, sent[i][1] - i тег в этом предложени
46          for sent in train_tokens_tags_list:
47              for i in range(len(sent)):
48                  B.loc[sent[i][0], sent[i][1]] += 1 # текущая i-пара слово-тег (обновите
49                  if len(sent) - 1 != i: # для последнего тега нет следующего тега
50                      A.loc[sent[i][1], sent[i + 1][1]] += 1 # пара тег-тег
51
52
53          # переходим к вероятностям
54
55          # нормируем по строке, то есть по всем всевозможным следующим тегам
56          A = A.divide(A.sum(axis=1), axis=0)
57
58          # нормируем по столбцу, то есть по всем всевозможным текущим словам
59          B = B / np.sum(B, axis=0)
60
61          self.A = A
62          self.B = B
63
64          return self
65
66
67      def predict(self, test_tokens_list):
68          """
69          test_tokens_list : массив предложений пар слово-тег (выборка для test)
70          """
71          predict_tags = OrderedDict({i : np.array([]) for i in range(len(test_tokens_lis
72
73          for i_sent in range(len(test_tokens_list)):
74
75              current_sent = [] # текущее предложение
76              len_sent = len(test_tokens_list[i_sent]) # длина предложения
77
78              q = np.zeros(shape=(len_sent + 1, len(self.tags)))
79              q[0] = 1 # нулевое состояние (равномерная инициализация по всем s)
80              back_point = np.zeros(shape=(len_sent + 1, len(self.tags))) # # argmax
81
```

```
            for t in range(len_sent):

                # если мы не встречали такое слово в обучении, то вместо него будет
                # самое популярное слово с самым популярным тегом (вопрос 2)
                current_word = test_tokens_list[i_sent][t]
                if current_word  not in self.words:
                    current_sent.append(self.most_popular_word_from_most_popular_tag)
                else:
                    current_sent.append(current_word)

                # через max выбираем следующий тег
                for i_s in range(len(self.tags)):

                    s = self.tags[i_s]

                    # формула (1)
                    q[t + 1][i_s] = np.max(q[t] *
                        self.A.loc[:, s] *
                        self.B.loc[current_sent[t], s])


                    # argmax формула(1)

                    # argmax, чтобы восстановить последовательность тегов
                    back_point[t + 1][i_s] = (q[t] * self.A.loc[:, s] *
                        self.B.loc[current_sent[t],s]).reset_index()[s].idxmax() # инде

            back_point = back_point.astype('int')

            # выписываем теги, меняя порядок на реальный
            back_tag = deque()
            current_tag = np.argmax(q[len_sent])
            for t in range(len_sent, 0, -1):
                back_tag.appendleft(self.tags[current_tag])
                current_tag = back_point[t, current_tag]

            predict_tags[i_sent] = np.array(back_tag)


        return predict_tags
```

Обучите скрытую марковскую модель:

```
my_model = HiddenMarkovModel()
my_model.fit(train_sents)
```

```
<__main__.HiddenMarkovModel at 0x7facc6546b50>
```

Проверьте работу реализованного алгоритма на следующих модельных примерах, проинтерпретируйте результат.

- 'He can stay'

- 'a cat and a dog'
- 'I have a television'
- 'My favourite character'

```
sents = [['He', 'can', 'stay'],
         ['a', 'cat', 'and', 'a', 'dog'],
         ['I', 'have', 'a', 'television'],
         ['My', 'favourite', 'character']]

preds = my_model.predict(sents)
preds
```

```
OrderedDict([(0, array(['NOUN', 'VERB', 'VERB'], dtype='<U4')),
             (1, array(['DET', 'NOUN', 'CONJ', 'DET', 'NOUN'], dtype='<U4')),
             (2, array(['NOUN', 'VERB', 'DET', 'NOUN'], dtype='<U4')),
             (3, array(['NOUN', 'NOUN', 'NOUN'], dtype='<U4'))])
```

Вопрос 3:

- Какой тег вы получили для слова `can`?

```
my_model.predict([['can']])
```

```
OrderedDict([(0, array(['VERB'], dtype='<U4'))])
```

Удивительно, если написать Can, то модель считает его существительным

Вопрос 4:

- Какой тег вы получили для слова `favourite`?

```
my_model.predict([['favourite']])
```

```
OrderedDict([(0, array(['NOUN'], dtype='<U4'))])
```

Примените модель к отложенной выборке Брауновского корпуса и подсчитайте точность определения тегов (accuracy). Сделайте выводы.

```
def accuracy_score(model, sents):
    true_pred = 0
    num_pred = 0
    i = 0

    for sent in sents:
        tags = np.array([tag for (word, tag) in sent])
        words = np.array([[word for (word, tag) in sent]])
        preds = np.array(model.predict(words)[0])
```

```
10
11
12          '''your code'''
13          match = (preds == tags)
14
15
16          true_pred += match.sum()
17          num_pred += len(match)
18      print("Accuracy:", true_pred / num_pred * 100, '%')
19      return (true_pred / num_pred)
```

```
1 accu = accuracy_score(my_model, test_sents)
```

```
Accuracy: 96.26470820500671 %
```

Вопрос 5:

- Какое качество вы получили(округлите до одного знака после запятой)?

```
1 np.round(accu, 1)
```

```
0.9
```

DefaultTagger

Вопрос 6:

- Какое качество вы бы получили, если бы предсказывали любой тег, как самый популярный тег на выборке train(округлите до одного знака после запятой)?

Вы можете испоьзовать DefaultTagger(метод tag для предсказания частей речи предложения)

```
1 from nltk.tag import DefaultTagger
2 default_tagger = DefaultTagger('NOUN')
```

```
1 def default_accuracy_score(model, sents):
2     true_pred = 0
3     num_pred = 0
4
5     for sent in sents:
6         tags = np.array([tag for (word, tag) in sent])
7
8         words = [word for (word, tag) in sent]
9         words_and_preds = model.tag(words)
10        preds = np.array([pred for (word, pred) in words_and_preds])
```

```
11
12         match = (tags == preds)
13
14         true_pred += match.sum()
15         num_pred += len(preds)
16
17     acc = true_pred / num_pred
18     print("Accuracy:", acc * 100, '%')
19     return acc
```

```
1 default_acc_score = default_accuracy_score(default_tagger, test_sents)
```

```
Accuracy: 20.217498764211566 %
```

```
1 np.round(default_acc_score, 1)
```

```
0.2
```

NLTK, Rnnmorph

Вспомним первый семинар нашего курса. В том семинаре мы с вами работали с некоторыми библиотеками.

Не забудьте преобразовать систему тэгов из `'en-ptb'` в `'universal'` с помощью функции `map_tag` или используйте `tagset='universal'`

```
1 pip install rnnmorph
```

```
Collecting rnnmorph
  Downloading rnnmorph-0.4.1.tar.gz (19.7 MB)
     |████████████████████████████████| 19.7 MB 6.7 MB/s
Requirement already satisfied: numpy>=1.12.1 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy>=0.19.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scikit-learn>=0.18.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: keras>=2.1.4 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: h5py>=2.7.0 in /usr/local/lib/python3.7/dist-packages
Collecting pymorphy2>=0.8
  Downloading pymorphy2-0.9.1-py3-none-any.whl (55 kB)
     |████████████████████████████████| 55 kB 3.6 MB/s
Collecting russian-tagsets==0.6
  Downloading russian-tagsets-0.6.tar.gz (23 kB)
Requirement already satisfied: tqdm>=4.14.0 in /usr/local/lib/python3.7/dist-packages
Collecting jsonpickle>=0.9.4
  Downloading jsonpickle-2.0.0-py2.py3-none-any.whl (37 kB)
Requirement already satisfied: nltk>=3.2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from nl
Requirement already satisfied: docopt>=0.6 in /usr/local/lib/python3.7/dist-packages
Collecting pymorphy2-dicts-ru<3.0,>=2.4
  Downloading pymorphy2_dicts_ru-2.4.417127.4579844-py2.py3-none-any.whl (8.2 MB)
     |████████████████████████████████| 8.2 MB 41.6 MB/s
```

```
Collecting dawg-python>=0.7.1
  Downloading DAWG_Python-0.7.2-py2.py3-none-any.whl (11 kB)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/d
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (f
Building wheels for collected packages: rnnmorph, russian-tagsets
  Building wheel for rnnmorph (setup.py) ... done
  Created wheel for rnnmorph: filename=rnnmorph-0.4.1-py3-none-any.whl size=19746379
  Stored in directory: /root/.cache/pip/wheels/b9/e8/8b/b4639d66ee5373f0db8dbe9a2ea94
  Building wheel for russian-tagsets (setup.py) ... done
  Created wheel for russian-tagsets: filename=russian_tagsets-0.6-py3-none-any.whl si
  Stored in directory: /root/.cache/pip/wheels/5e/b4/26/9c17a7cdcfc6b8cf43111312f3e7d
Successfully built rnnmorph russian-tagsets
Installing collected packages: pymorphy2-dicts-ru, dawg-python, russian-tagsets, pymo
Successfully installed dawg-python-0.7.2 jsonpickle-2.0.0 pymorphy2-0.9.1 pymorphy2-d
```

```
1 from nltk.tag.mapping import map_tag
```

```
1 import nltk
2 nltk.download('averaged_perceptron_tagger')
3 # nltk.pos_tag(..., tagset='universal')
4
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
1 from rnnmorph.predictor import RNNMorphPredictor
2 predictor = RNNMorphPredictor(language="en")
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!
WARNING:tensorflow:Layer LSTM_1_forward will not use cuDNN kernels since it doesn't m
WARNING:tensorflow:Layer LSTM_1_backward will not use cuDNN kernels since it doesn't
WARNING:tensorflow:Layer LSTM_0 will not use cuDNN kernels since it doesn't meet the
WARNING:tensorflow:Layer LSTM_0 will not use cuDNN kernels since it doesn't meet the
WARNING:tensorflow:Layer LSTM_0 will not use cuDNN kernels since it doesn't meet the
```

```
1 # predictor.predict(np.array(['Hello' 'sadaf']))
2 predictor.predict(['I', 'bear', 'a', 'bag'])[2].pos
```

```
'DET'
```

Вопрос 7:

- Какое качество вы получили, используя каждую из двух библиотек? Сравните их результаты.
- Качество с библиотекой rnnmorph должно быть хуже, так как там используется немного другая система тэгов. Какие здесь отличия?

```
def nltk_accuracy_score(tagset, sents):
    true_pred = 0
    num_pred = 0

    for sent in sents:
        tags = np.array([tag for (word, tag) in sent])

        words = [word for (word, tag) in sent]
        words_and_preds = nltk.pos_tag(words, tagset=tagset)
        preds = np.array([pred for (word, pred) in words_and_preds])
        # print(preds)

        match = (tags == preds)
        #print(match)

        true_pred += match.sum()
        num_pred += len(preds)

    acc = true_pred / num_pred
    print("Accuracy:", np.round(acc * 100, 1), '%')
    return acc
```

```
nltk_acc_score = nltk_accuracy_score('universal', test_sents)
nltk_acc_score
```

```
Accuracy: 89.2 %
0.8922392486406328
```

```
def accuracy_score_r(model, sents):
    true_pred = 0
    num_pred = 0

    for sent in sents:
        tags = np.array([tag for (word, tag) in sent])
        words = [word for (word, tag) in sent]

        full_preds = predictor.predict(words)

        preds = np.array([full_preds[i].pos for i in range(len(full_preds))])


        # preds = np.array(list(model.predict(word)[0].pos for word in words))

        # print(zip(preds[:10], tags[:10]))
        # for (k, m) in zip(preds[:10], tags[:10]):
```

```
        #     print(k, m)

        '''your code'''
        match = (preds == tags)


        true_pred += match.sum()
        num_pred += len(match)
    print("Accuracy:", true_pred / num_pred * 100, '%')
    return (true_pred / num_pred)
```

```
accuracy_score_r(predictor, test_sents)
```

```
Accuracy: 62.827483934750376 %
0.6282748393475037
```

BiLSTMTagger

Подготовка данных

Изменим структуру данных

```
pos_data = [list(zip(*sent)) for sent in brown_tagged_sents]
print(pos_data[0])
```

```
[('The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation
```

До этого мы писали много кода сами, теперь пора эксплуатировать pytorch

```
from torchtext.legacy.data import Field, BucketIterator
import torchtext

# наши поля
WORD = Field(lower=True)
TAG = Field(unk_token=None) # все токены нам извсетны

# создаем примеры
examples = []
for words, tags in pos_data:
    examples.append(torchtext.legacy.data.Example.fromlist([list(words), list(tags)], f
```

Вот один наш пример:

```
print(vars(examples[0]))
```

```
{'words': ['the', 'fulton', 'county', 'grand', 'jury', 'said', 'friday', 'an', 'inves
```

Теперь формируем наш датасет

```
# кладем примеры в наш датасет
dataset = torchtext.legacy.data.Dataset(examples, fields=[('words', WORD), ('tags', TAG

train_data, valid_data, test_data = dataset.split(split_ratio=[0.8, 0.1, 0.1])

print(f"Number of training examples: {len(train_data.examples)}")
print(f"Number of validation examples: {len(valid_data.examples)}")
print(f"Number of testing examples: {len(test_data.examples)}")
```

```
Number of training examples: 45872
Number of validation examples: 5734
Number of testing examples: 5734
```

Построим словари. Параметр `min_freq` выберете сами. При построении словаря используем только **train**

```
WORD.build_vocab(train_data, min_freq=10)
TAG.build_vocab(train_data)

print(f"Unique tokens in source (ru) vocabulary: {len(WORD.vocab)}")
print(f"Unique tokens in target (en) vocabulary: {len(TAG.vocab)}")

print(WORD.vocab.itos[::200])
print(TAG.vocab.itos)
```

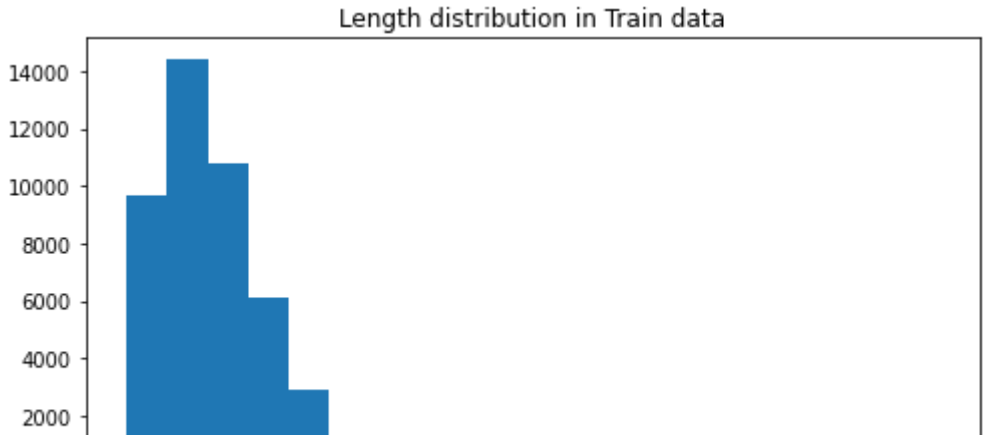
```
Unique tokens in source (ru) vocabulary: 7325
Unique tokens in target (en) vocabulary: 13
['<unk>', 'away', 'close', 'bring', 'southern', 'leaders', 'units', 'robert', 'signs
['<pad>', 'NOUN', 'VERB', '.', 'ADP', 'DET', 'ADJ', 'ADV', 'PRON', 'CONJ', 'PRT', 'NU
```

```
print(vars(train_data.examples[9]))
```

```
{'words': ['the', "women's", 'faces', 'had', 'hardened', 'after', 'my', 'statement',
```

Посмотрим с насколько большими предложениями мы имеем дело

```
length = map(len, [vars(x)['words'] for x in train_data.examples])

plt.figure(figsize=[8, 4])
plt.title("Length distribution in Train data")
plt.hist(list(length), bins=20);
```



Для обучения `BiLSTM` лучше использовать colab

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

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

```
device(type='cuda')
```

Для более быстрого и устойчивого обучения сгруппируем наши данные по батчам

```
# бьем нашу выборку на батч, не забывая сначала отсортировать выборку по длине
def _len_sort_key(x):
    return len(x.words)

BATCH_SIZE = 128

train_iterator, valid_iterator, test_iterator = BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = BATCH_SIZE,
    device = device,
    sort_key=_len_sort_key
)
```

```
# посморим  на количество батчей
list(map(len, [train_iterator, valid_iterator, test_iterator]))
```

```
[359, 45, 45]
```

Модель и её обучение

Инициализируем нашу модель

```
class LSTMTagger(nn.Module):

    def __init__(self, input_dim, emb_dim, hid_dim, output_dim, dropout, bidirectional=
        super().__init__()


        self.embeddings = nn.Embedding(num_embeddings=input_dim, embedding_dim=emb_dim)
        self.dropout = nn.Dropout(p=dropout)

        self.rnn = nn.LSTM(input_size=emb_dim, hidden_size=hid_dim,
                           dropout=dropout, bidirectional=bidirectional)
        # если bidirectional, то предсказываем на основе конкатенации двух hidden
        self.tag = nn.Linear((1 + bidirectional) * hid_dim, output_dim)

    def forward(self, sent):

        #sent = [sent len, batch size]

        # не забываем применить dropout к embedding
        embedded =self.embeddings(sent)

        output, _ = self.rnn(self.dropout(embedded))
        #output = [sent len, batch size, hid dim * n directions]

        prediction = self.tag(output)

        return prediction

# параметры модели
INPUT_DIM = len(WORD.vocab) # количество эмбеддингов
OUTPUT_DIM = len(TAG.vocab) # количетво классифицируемых тегов
EMB_DIM = 300
HID_DIM = 16
DROPOUT = 0.2
BIDIRECTIONAL =True

model = LSTMTagger(input_dim=INPUT_DIM, emb_dim=EMB_DIM,
                   hid_dim=HID_DIM, output_dim=OUTPUT_DIM,
                   dropout=DROPOUT, bidirectional=BIDIRECTIONAL).to(device)

# инициализируем веса
def init_weights(m):
    for name, param in m.named_parameters():
        nn.init.uniform_(param, -0.08, 0.08)

model.apply(init_weights)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/rnn.py:65: UserWarning: dropo
  "num_layers={}".format(dropout, num_layers))
LSTMTagger(
  (embeddings): Embedding(7325, 300)
  (dropout): Dropout(p=0.2, inplace=False)
  (rnn): LSTM(300, 16, dropout=0.2, bidirectional=True)
  (tag): Linear(in_features=32, out_features=13, bias=True)
)
```

Подсчитаем количество обучаемых параметров нашей модели

```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters())

print(f'The model has {count_parameters(model):,} trainable parameters')
```

```
The model has 2,238,633 trainable parameters
```

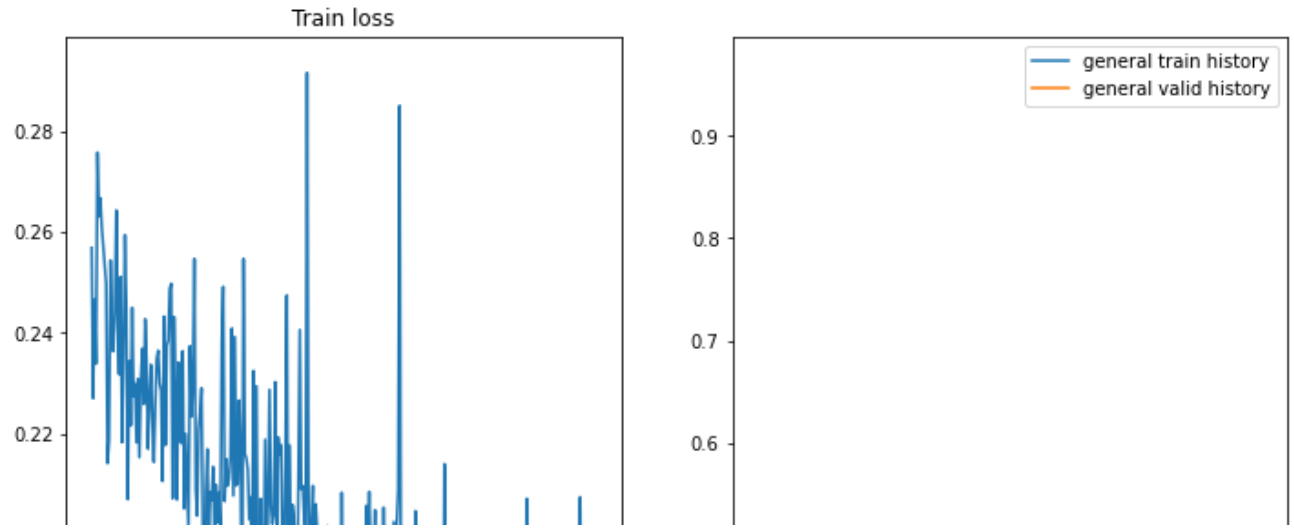
Погнали обучать

```
PAD_IDX = TAG.vocab.stoi['<pad>']
optimizer = optim.Adam(model.parameters())
criterion = nn.CrossEntropyLoss(ignore_index = PAD_IDX)

def train(model, iterator, optimizer, criterion, clip, train_history=None, valid_histor
    model.train()

    epoch_loss = 0
    history = []
    for i, batch in enumerate(iterator):
        '''your code'''
        words = batch.words.to(device)
        tags  = batch.tags.to(device)

        optimizer.zero_grad()

        output = model(words)

        #tags = [sent len, batch size]
        #output = [sent len, batch size, output dim]

        output = output.view(-1, output.shape[2])
        tags = tags.view(-1)

        #tags = [sent len * batch size]
        #output = [sent len * batch size, output dim]

        loss = criterion(output, tags)

        loss.backward()

        # Gradient clipping(решение проблемы взрыва граденты), clip - максимальная норм
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=clip)

        optimizer.step()

        epoch_loss += loss.item()

        history.append(loss.cpu().data.numpy())
        if (i+1)%10==0:
```

```
            fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12, 8))

            clear_output(True)
            ax[0].plot(history, label='train loss')
            ax[0].set_xlabel('Batch')
            ax[0].set_title('Train loss')

            if train_history is not None:
                ax[1].plot(train_history, label='general train history')
                ax[1].set_xlabel('Epoch')
            if valid_history is not None:
                ax[1].plot(valid_history, label='general valid history')
            plt.legend()

            plt.show()


    return epoch_loss / len(iterator)

def evaluate(model, iterator, criterion):
    model.eval()

    epoch_loss = 0

    history = []

    with torch.no_grad():

        for i, batch in enumerate(iterator):

            '''your code'''
            words = batch.words.to(device)
            tags  = batch.tags.to(device)

            output = model(words)

            #tags = [sent len, batch size]
            #output = [sent len, batch size, output dim]

            output = output.view(-1, output.shape[2])
            tags = tags.view(-1)

            #tags = [sent len * batch size]
            #output = [sent len * batch size, output dim]

            loss = criterion(output, tags)

            epoch_loss += loss.item()

    return epoch_loss / len(iterator)

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
```

```
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

```
import time
import math
import matplotlib
matplotlib.rcParams.update({'figure.figsize': (16, 12), 'font.size': 14})
import matplotlib.pyplot as plt
%matplotlib inline
from IPython.display import clear_output

train_history = []
valid_history = []

N_EPOCHS = 2
CLIP = 1

best_valid_loss = float('inf')

for epoch in range(N_EPOCHS):

    start_time = time.time()

    train_loss = train(model, train_iterator, optimizer, criterion, CLIP, train_history
    valid_loss = evaluate(model, valid_iterator, criterion)

    end_time = time.time()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'best-val-model.pt')

    train_history.append(train_loss)
    valid_history.append(valid_loss)
    print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train PPL: {math.exp(train_loss):7.3f}')
    print(f'\t Val. Loss: {valid_loss:.3f} |  Val. PPL: {math.exp(valid_loss):7.3f}')
```



Применение модели

```
def accuracy_model(model, iterator):
    model.eval()

    true_pred = 0
    num_pred = 0

    with torch.no_grad():
        for i, batch in enumerate(iterator):
            '''your code'''
            words = batch.words.to(device)
            tags  = batch.tags.to(device)

            output = model(words)

            #output = [sent len, batch size, output dim]
            output = torch.argmax(output, dim=2)

            #output = [sent len, batch size]
            predict_tags = output.cpu().numpy()
            true_tags = tags.cpu().numpy()

            true_pred += np.sum((true_tags == predict_tags) & (true_tags != PAD_IDX))
            num_pred += np.prod(true_tags.shape) - (true_tags == PAD_IDX).sum()

    return round(true_pred / num_pred * 100, 3)
```

```
print("Accuracy:", accuracy_model(model, test_iterator), '%')
```

```
Accuracy: 95.707 %
```


```
print("Accuracy:", accuracy_model(model, test_iterator), '%')
```

```
Accuracy: 95.707 %
```

Вы можете улучшить качество, изменяя параметры модели. Но чтобы добиться нужного качества, вам неообходимо взять все выборку, а не только категорию `humor`.

```
# brown_tagged_sents = brown.tagged_sents(tagset="universal")
```

Вам неоходимо добиться качества не меньше, чем `accuracy = 93 %`

```
best_model = LSTMTagger(INPUT_DIM, EMB_DIM, HID_DIM, OUTPUT_DIM, DROPOUT, BIDIRECTIONAL
best_model.load_state_dict(torch.load('best-val-model.pt'))
assert accuracy_model(best_model, test_iterator) >= 93
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/rnn.py:65: UserWarning: dropo
  "num_layers={}".format(dropout, num_layers))
```

Пример решение нашей задачи:

```
def print_tags(model, data):
    model.eval()

    with torch.no_grad():
        words, _ = data
        example = torch.LongTensor([WORD.vocab.stoi[elem] for elem in words]).unsqueeze

        output = model(example).argmax(dim=-1).cpu().numpy()
        tags = [TAG.vocab.itos[int(elem)] for elem in output]

        for token, tag in zip(words, tags):
            print(f'{token:15s}{tag}')
```

```
print_tags(model, pos_data[-1])
```

```
From            VERB
what            DET
I               NOUN
was             VERB
able            ADJ
to              PRT
gauge           VERB
in              ADP
a               DET
swift           NOUN
,               .
greedy          ADJ
glance          NOUN
,               .
the             DET
figure          NOUN
inside          ADP
the             DET
coral-colored   NOUN
```

```
boucle          NOUN
dress           NOUN
was             VERB
stupefying      ADJ
.               .
```

Сравните результаты моделей HiddenMarkov, LstmTagger:

- при обучение на маленькой части корпуса, например, на категории humor
- при обучении на всем корпусе

HiddenMarkov

На категории humor 88 %
На всем корпусе 96 %(но считалось целый час)

LstmTagger

На категории humor - 86 %
На всем корпусе - 95 %

```
1
```