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).

```
import nltk
import pandas as pd
import numpy as np
from collections import OrderedDict, deque
from nltk.corpus import brown
import matplotlib.pyplot as plt
```

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

Загрузим brown корпус

```
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

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

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

Мы имеем массив предложений пар (слово-тег)

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

```
[[('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'), ('.', '.')],
[('The', 'DET'), ('jury', 'NOUN'), ('further', 'ADV'), ('said', 'VERB'), ('in',
'ADP'), ('term-end', 'NOUN'), ('presentments', 'NOUN'), ('that', 'ADP'), ('the',
'DET'), ('City', 'NOUN'), ('Executive', 'ADJ'), ('Committee', 'NOUN'), (',', '.'),
('which', 'DET'), ('had', 'VERB'), ('over-all', 'ADJ'), ('charge', 'NOUN'), ('of',
'ADP'), ('the', 'DET'), ('election', 'NOUN'), (',', '.'), ('``', '.'), ('deserves',
'VERB'), ('the', 'DET'), ('praise', 'NOUN'), ('and', 'CONJ'), ('thanks', 'NOUN'),
('of', 'ADP'), ('the', 'DET'), ('City', 'NOUN'), ('of', 'ADP'), ('Atlanta',
'NOUN'), ("''", '.'), ('for', 'ADP'), ('the', 'DET'), ('manner', 'NOUN'), ('in',
```

```
     'ADP'), ('which', 'DET'), ('the', 'DET'), ('election', 'NOUN'), ('was', 'VERB'),
     ('conducted', 'VERB'), ('.', '.')], ...]
```

Первое предложение

```
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)

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

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

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

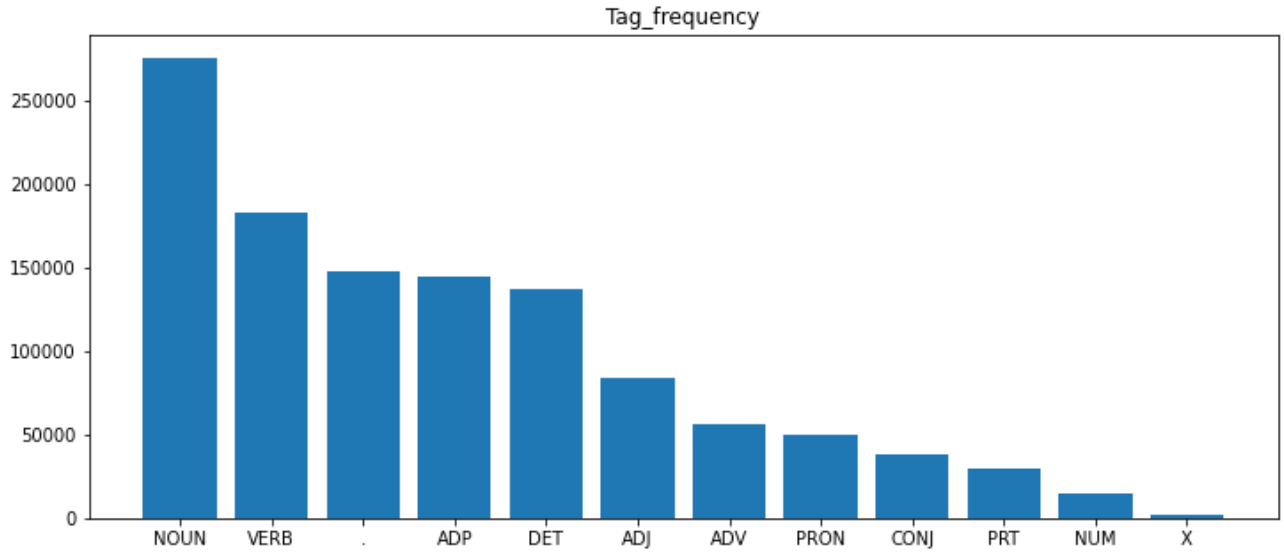
```
tag_num = pd.Series(nltk.FreqDist(tags)).sort_values(ascending=False) # тег - кол-во тега
word_num = pd.Series(nltk.FreqDist(words)).sort_values(ascending=False) # слово - кол-во с
```

```
Кол-во предложений:  57340
```

```
tag_num[:5]
```

```
NOUN    275558
VERB    182750
.       147565
ADP     144766
DET     137019
dtype: int64
```

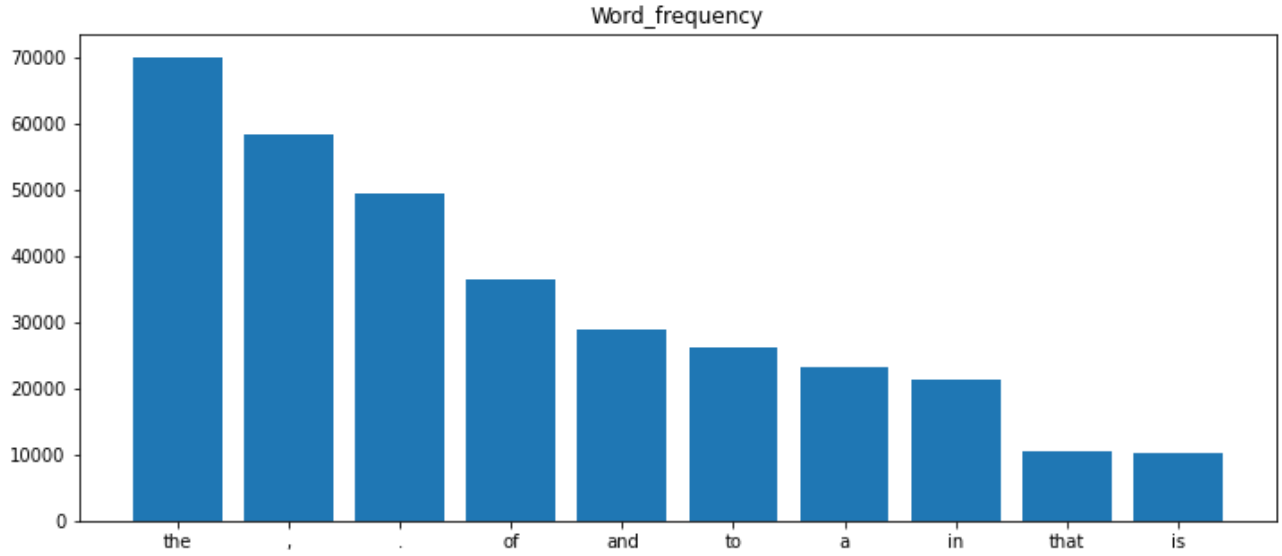
```
plt.figure(figsize=(12, 5))
plt.bar(tag_num.index, tag_num.values)
plt.title("Tag_frequency")
plt.show()
```




```
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*)

```
polupar_tag = 'NOUN'
words_polular_tag = [word for (word, tag) in brown_tagged_words if tag == polupar_tag]
the_popular_word = nltk.FreqDist(words_polular_tag).max()
the_popular_word
```

```
'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')
# 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, train_size=0.9, test_si
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: VisibleDeprecationWa
```

```
len(train_sents)
```

```
947
```

```
len(test_sents)
```

```
106
```

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

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

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]

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

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

    A = pd.DataFrame({'{}'.format(tag) : [0] * len(tag_num) for tag in tag_num.index},
    B = pd.DataFrame({'{}'.format(tag) : [0] * len(word_num) for tag in tag_num.index}

    # Вычисляем матрицу A и B по частотам слов и тегов

    # sent - предложение
    # sent[i][0] - i слово в этом предложении, sent[i][1] - i тег в этом предложении
    for sent in train_tokens_tags_list:
        for i in range(len(sent)):
            B.loc[sent[i][0], sent[i][1]] += 1 # текущая i-пара слово-тег (обновите ма
            if len(sent) - 1 != i: # для последнего тега нет следующего тега
                A.loc[sent[i][1], sent[i + 1][1]] += 1 # пара тег-тег


    # переходим к вероятностям

    # нормируем по строке, то есть по всем всевозможным следующим тегам
    A = A.divide(A.sum(axis=1), axis=0)

    # нормируем по столбцу, то есть по всем всевозможным текущим словам
    B = B / np.sum(B, axis=0)

    self.A = A
    self.B = B

    return self


def predict(self, test_tokens_list):
    """
    test_tokens_list : массив предложений пар слово-тег (выборка для test)
    """
    predict_tags = OrderedDict({i : np.array([]) for i in range(len(test_tokens_list))

    for i_sent in range(len(test_tokens_list)):

        current_sent = test_tokens_list[i_sent] # текущее предложение
        len_sent = len(current_sent) # длина предложения

        q = np.zeros(shape=(len_sent + 1, len(self.tags)))
        q[0] = 1 # нулевое состояние (равномерная инициализация по всем s)
        back_point = np.zeros(shape=(len_sent + 1, len(self.tags))) # # argmax
```

```
            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)
                if current_sent[t] not in self.words:
                    current_sent[t] = 'time'

                # 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() # индекс

            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:

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

```
<__main__.HiddenMarkovModel at 0x7fde46818450>
```

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', 'telev
         ['My', 'favourite', 'character']]
predictions = model.predict(sents)
predictions
```

```
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`?

```
predictions[0][1]
```

```
'VERB'
```

Вопрос 4:

- Какой тег вы получили для слова `favourite`?

```
predictions[3][1]
```

```
'NOUN'
```

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

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

        predictions = model([words])[0]

        true_pred += sum([tag == pred for (tag, pred) in zip(tags, predictions)])
        num_pred +=len(words)
    print("Accuracy:", true_pred / num_pred * 100, '%')
```

```
accuracy_score(model.predict, test_sents)
```

```
Accuracy: 89.35762224352828 %
```

Вопрос 5:

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

```
round(89.35762224352828, 1)
```

```
    89.4
```

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)

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

```
accuracy_score(default_tagger.tag, train_sents)
```

```
    Accuracy: 0.8822479473711051 %
```

```
round(0.882247947371105, 1)
```

```
    0.9
```

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'`

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

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

```
    num_pred = 0

    for sent in sents:
        tags = [tag for (word, tag) in sent]
        words = [word for (word, tag) in sent]
        predictions = model(words)
        preds_tag = [map_tag('en-ptb',·'universal',·tag) for word, tag in predictions]
        true_pred += sum([tag == pred for (tag, pred) in zip(tags, preds_tag)])
        num_pred +=len(preds_tag)
    print("Accuracy:", true_pred / num_pred * 100, '%')


import nltk
nltk.download('averaged_perceptron_tagger')


accuracy_score_nltk(nltk.pos_tag, train_sents)

     Accuracy: 89.20903666683667 %


!pip3 install rnnmorph -q


nltk.download('omw-1.4')

     [nltk_data] Downloading package omw-1.4 to /root/nltk_data...
     [nltk_data]   Package omw-1.4 is already up-to-date!
     True


def accuracy_score_rnnmorph(model, sents):
    true_pred = 0
    num_pred = 0

    for sent in sents:
        tags = [tag for (word, tag) in sent]
        words = [word for (word, tag) in sent]
        predictions = model(words)
        preds_tag = [pred.pos for pred in predictions]
        true_pred += sum([tag == pred for (tag, pred) in zip(tags, preds_tag)])
        num_pred +=len(preds_tag)
    print("Accuracy:", true_pred / num_pred * 100, '%')


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

     [nltk_data] Downloading package wordnet to /root/nltk_data...
     [nltk_data]   Package wordnet is already up-to-date!
     [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
     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
```

```
accuracy_score_rnnmorph(predictor.predict, train_sents)
```

```
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 56ms/step
1/1 [==============================] - 0s 26ms/step
```

```
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 80ms/step
```

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?

```
print(f'Accuracy by NLTK {round(89.20903666683667, 1)}')
print(f'Accuracy by RNNMorphPredictor {round(63.55245040542607, 1)}')
```

```
Accuracy by NLTK 89.2
Accuracy by RNNMorphPredictor 63.6
```

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

```
!pip install torchtext==0.10.0 -q
```

```
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)], fie
```

Вот один наш пример:

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

```
{'words': ['it', 'was', 'among', 'these', 'that', 'hinkle', 'identified', 'a', 'phot
```

Теперь формируем наш датасет

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

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=2)
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)
```

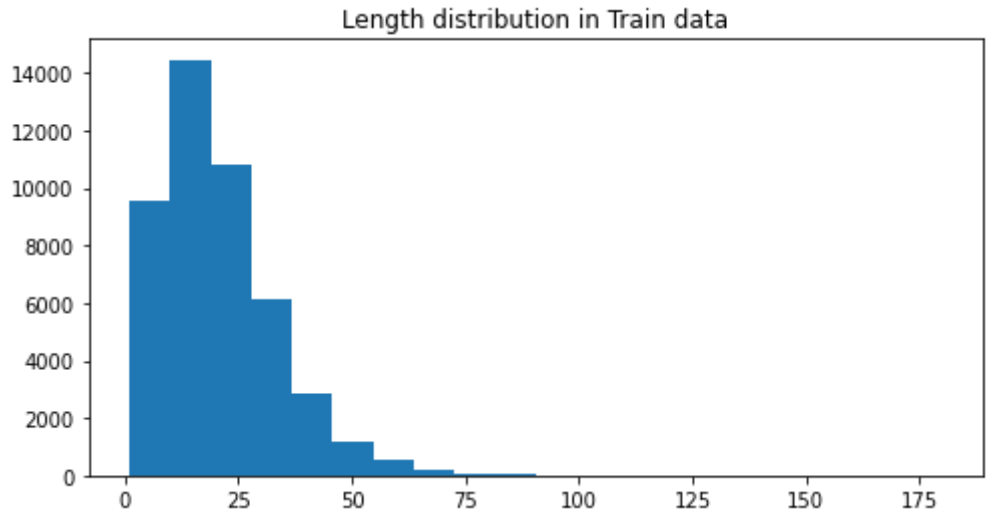
```
Unique tokens in source (ru) vocabulary: 24759
Unique tokens in target (en) vocabulary: 13
['<unk>', '2', 'became', 'growth', 'fear', 'letters', 'someone', 'requirements', 'gr
['<pad>', 'NOUN', 'VERB', '.', 'ADP', 'DET', 'ADJ', 'ADV', 'PRON', 'CONJ', 'PRT', 'N
```

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

```
{'words': ['richardson', 'wondered', 'when', 'it', 'would', 'be', 'unloaded', '.'],
```

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 = 32

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]))
```

```
[1434, 180, 180]
```

Модель и её обучение

Инициализируем нашу модель

```
class LSTMTagger(nn.Module):

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


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

        self.rnn = nn.LSTM(input_size=emb_dim, hidden_size=hid_dim, dropout=dropout, bidir
        # 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.dropout(self.embeddings(sent))

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

        prediction = self.tag(output)

        return prediction

# параметры модели
INPUT_DIM = len(WORD.vocab)
OUTPUT_DIM = len(TAG.vocab)
EMB_DIM = 300
HID_DIM = 600
DROPOUT = 0.2
BIDIRECTIONAL = True

model = LSTMTagger(input_dim=INPUT_DIM, emb_dim=EMB_DIM, hid_dim=HID_DIM, output_dim=OUTPU

# 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)
```

```
 LSTMTagger(
   (embeddings): Embedding(24759, 300)
   (dropout): Dropout(p=0.2, inplace=False)
```

```
    (rnn): LSTM(300, 600, dropout=0.2, bidirectional=True)
    (tag): Linear(in_features=1200, out_features=13, bias=True)
  )
```

Подсчитаем количество обучаемых параметров нашей модели

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

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

```
The model has 11,772,913 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_history=N
  model.train()

  epoch_loss = 0
  history = []
  for i, batch in enumerate(iterator):

    words = batch.words
    tags = batch.tags

    optimizer.zero_grad()

    output = model(words)

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

    output = output.view(-1, output.size(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 норма векто
    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):

            words = batch.words
            tags = batch.tags

            output = model(words)


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

            output = output.view(-1, output.size(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
```

Ошибка, так как цикл обучения был приостановлен ранее количества эпох, так как модель начинала переобучаться

```
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 = 10
CLIP = 5

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, v
    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}')
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-239-abeedbac37c0> in <module>
     19     start_time = time.time()
     20
---> 21     train_loss = train(model, train_iterator, optimizer, criterion, CLIP,
train_history, valid_history)
     22     valid_loss = evaluate(model, valid_iterator, criterion)
     23

                                 13 frames
<decorator-gen-2> in __call__(self, obj)
```

Применение модели

```
    739         """
def accuracy_model(model, iterator):

  model.eval()

  true_pred = 0
  num_pred = 0

  with torch.no_grad():
      for i, batch in enumerate(iterator):
        words = batch.words
        tags = batch.tags

        output = model(words)

        #output = [sent len, batch size, output dim]
        output = torch.argmax(output, dim=-1)

        #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: 97.449 %
```

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).t
best_model.load_state_dict(torch.load('best-val-model_full_corpus.pt'))
acc = accuracy_model(best_model, test_iterator)
print(acc)
assert acc >= 93
```

```
97.592
```

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(1)

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

Сравните результаты моделей HiddenMarkov, LstmTagger:

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

Обучение на категории humor

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

train_sents, test_sents = train_test_split( my_brown_tagged_sents, train_size=0.9, test_si
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: VisibleDeprecationWa
  """
```

HiddenMarkov

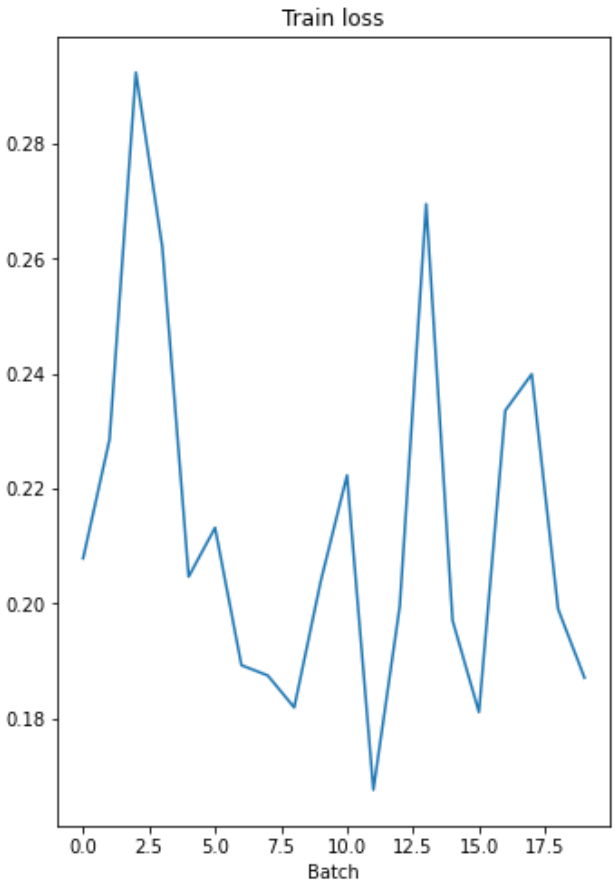
```
model = HiddenMarkovModel()
model.fit(train_sents)

accuracy_score(model.predict, test_sents)
```

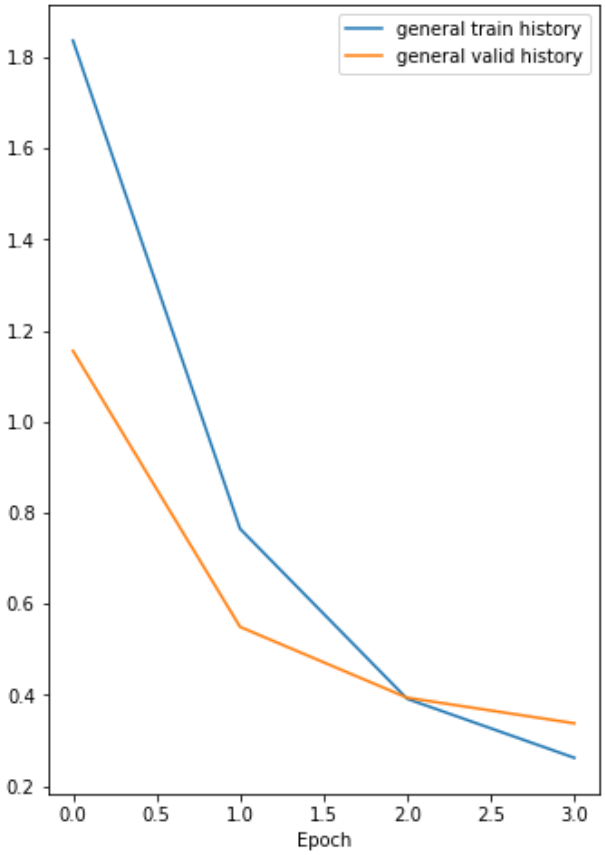
```
Accuracy: 89.35762224352828 %
```

LstmTagger

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

WORD = Field(lower=True)
TAG = Field(unk_token=None)

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

dataset = torchtext.legacy.data.Dataset(examples=examples, fields=[('words', WORD), ('tags

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

WORD.build_vocab(train_data, min_freq=2)
TAG.build_vocab(train_data)

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

```
BATCH_SIZE = 32

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
)

INPUT_DIM = len(WORD.vocab)
OUTPUT_DIM = len(TAG.vocab)
EMB_DIM = 300
HID_DIM = 600
DROPOUT = 0.2
BIDIRECTIONAL = True

model = LSTMTagger(input_dim=INPUT_DIM, emb_dim=EMB_DIM, hid_dim=HID_DIM, output_dim=OUTPU
model.apply(init_weights)

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

train_history = []
valid_history = []

N_EPOCHS = 5
CLIP = 5

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, v
    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_humor.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}')
```






```
Epoch: 05 | Time: 0m 1s
        Train Loss: 0.208 | Train PPL:   1.231
         Val. Loss: 0.320 |  Val. PPL:   1.377
```

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

```
Accuracy: 90.049 %
```

Обучение на всем корпусе

[ ] ↳ *Скрыто 6 ячеек.*

Платные продукты Colab - Отменить подписку

✓ 0 сек. выполнено в 20:03