Тема “Классификация текста”

Взять ноутбук colab_text_classification_part1.ipynb который разбирали на занятии и добавить пункты которые мы пропустили

1. Посмотрите на токены если будут мусорные добавьте их в стоп слова и обучите заново
2. Проверьте изменилось ли качество при лемматизации/и без неё
3. Замените все токены которые принадлежат сущностям на их тег. Проверьте изменилось ли качество после этого

B [1]:
```
# !wget -O imdb.zip -qq --no-check-certificate "https://drive.google.com/uc?export=download&id=1vrQ5czMHoO3pEnmofFskym
# !unzip imdb.zip
# !pip -q install eli5
# !pip -q install spacy
# !python -m spacy download en
```

```
Archive:  imdb.zip
  inflating: test.tsv
  inflating: train.tsv
     |████████████████████████████████| 216 kB 8.3 MB/s
     |████████████████████████████████| 133 kB 57.7 MB/s
  Building wheel for eli5 (setup.py) ... done
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behav
iour is the source of the following dependency conflicts.
flask 1.1.4 requires Jinja2<3.0,>=2.10.1, but you have jinja2 3.1.2 which is incompatible.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatible.
Collecting en_core_web_sm==2.2.5
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.2.5/en_core_web_sm-2.2.5.t
ar.gz (https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.2.5/en_core_web_sm-2.2.5.tar.gz)
(12.0 MB)
     |████████████████████████████████| 12.0 MB 6.8 MB/s
Requirement already satisfied: spacy>=2.2.2 in /usr/local/lib/python3.7/dist-packages (from en_core_web_sm==2.2.5)
(2.2.4)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2
->en_core_web_sm==2.2.5) (1.0.7)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->en_core_web_s
m==2.2.5) (57.4.0)
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->en_core_web
_sm==2.2.5) (7.4.0)
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->en_
core_web_sm==2.2.5) (0.9.1)
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->en_c
ore_web_sm==2.2.5) (1.0.5)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->
en_core_web_sm==2.2.5) (2.23.0)
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->
en_core_web_sm==2.2.5) (1.0.0)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->en
_core_web_sm==2.2.5) (3.0.6)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->en_c
ore_web_sm==2.2.5) (4.64.0)
Requirement already satisfied: blis<0.5.0,>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->en_co
re_web_sm==2.2.5) (0.4.1)
Requirement already satisfied: plac<1.2.0,>=0.9.6 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->en_co
re_web_sm==2.2.5) (1.1.3)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->en_core_we
b_sm==2.2.5) (1.21.6)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy>=2.2.2->en_c
ore_web_sm==2.2.5) (2.0.6)
Requirement already satisfied: importlib-metadata>=0.20 in /usr/local/lib/python3.7/dist-packages (from catalogue<1.
1.0,>=0.0.7->spacy>=2.2.2->en_core_web_sm==2.2.5) (4.11.3)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/dist-packages (from importlib-met
adata>=0.20->catalogue<1.1.0,>=0.0.7->spacy>=2.2.2->en_core_web_sm==2.2.5) (4.2.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=0.20->ca
talogue<1.1.0,>=0.0.7->spacy>=2.2.2->en_core_web_sm==2.2.5) (3.8.0)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.1
3.0->spacy>=2.2.2->en_core_web_sm==2.2.5) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (fro
m requests<3.0.0,>=2.13.0->spacy>=2.2.2->en_core_web_sm==2.2.5) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->
spacy>=2.2.2->en_core_web_sm==2.2.5) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.
13.0->spacy>=2.2.2->en_core_web_sm==2.2.5) (2021.10.8)
✓ Download and installation successful
You can now load the model via spacy.load('en_core_web_sm')
✓ Linking successful
/usr/local/lib/python3.7/dist-packages/en_core_web_sm -->
/usr/local/lib/python3.7/dist-packages/spacy/data/en
You can now load the model via spacy.load('en')
```

Классификация текстов

Начнём с самого простого - анализа тональности текста.

Будем классифицировать отзывы с IMDB на положительные/отрицательные.

Датасет взят с http://ai.stanford.edu/~amaas/data/sentiment/ (http://ai.stanford.edu/~amaas/data/sentiment/)

В [3]:

```
is_positive     review
0       "Dreamgirls, despite its fistful of Tony wins in an incredibly weak year on Broadway, has never been what one
would call a jewel in the crown of stage musicals. However, that is not to say that in the right cinematic hands it c
ould not be fleshed out and polished into something worthwhile on-screen. Unfortunately, what transfers to the screen
is basically a slavishly faithful version of the stage hit with all of its inherent weaknesses intact. First, the sco
re has never been one of the strong points of this production and the film does not change that factor. There are lot
s of songs (perhaps too many?), but few of them are especially memorable. The closest any come to catchy tunes are th
e title song and One Night Only - the much acclaimed And I Am Telling You That I Am Not Going is less a great song th
an it is a dramatic set piece for the character of Effie (Jennifer Hudson). The film is slick and technically well-pr
oduced, but the story and characters are surprisingly thin and lacking in any resonance. There is some interest in th
e opening moments, watching Jamie Foxx's Svengali-like manager manipulate his acts to the top, but that takes a back
seat in the latter portion of the film, when the story conveniently tries to cast him as a villain, despite his havin
g been right from a business stand-point for a good majority of the film. Beyonce Knowles is lovely and sings her son
gs perfectly well, but is stuck with a character who is basically all surface glitz. Anika Noni Rose as the third mem
ber of the Dreamgirls trio literally has nothing to do for the entire film. Eddie Murphy acquits himself well as a si
nger obviously based on James Brown, but the role is not especially meaty and ultimately has little impact. Foxx woul
d seem ideal casting, but he seems oddly withdrawn and bored. The film's biggest selling point is surely former Ameri
can Idol contestant/Oscar winner Jennifer Hudson in the central role of Effie White, the temperamental singer who get
s booted from the group and makes a triumphant closing act return. For me, Effie has always been a big problem in bot
```

В [1]:

```
import pandas as pd

# Сброс ограничений на количество выводимых рядов
# pd.set_option('display.max_rows', None)

# Сброс ограничений на число столбцов
# pd.set_option('display.max_columns', None)

# Сброс ограничений на количество символов в записи
```

В [2]:

```
train_df = pd.read_csv("data/train.tsv", delimiter="\t")
test_df = pd.read_csv("data/test.tsv", delimiter="\t")

print('Train size = {}'.format(len(train_df)))
```

```
Train size = 25000
Test size = 25000
```

Посмотрите глазами на тексты? Какие есть зацепки, как определить, что это за сентимент?

Самое простое, как всегда - найти ключевые слова.

В [3]:

```
#@title Начинаем классифицировать! { vertical-output: true, display-mode: "form" }
positive_words = 'love', 'great', 'best', 'wonderful' #@param {type:"raw"}
negative_words = 'worst', 'awful', '1/10', 'crap' #@param {type:"raw"}
```

```
positive_words = ('love',  # 'любовь',
'10/10',
'9/10',
'8/10',
'7/10',
'great',  # 'большой',
'best',  # 'Лучший',
'wonderful',  # 'замечательно',
'amazing',  #'удивительно',
'powerful',  # 'мощный',
'excellent',  # 'превосходно',
'perfect',  # 'идеально',
'ideal',  # 'идеальный',
'brilliant',  # 'блестящий',
'incredible',  # 'невероятный',
'outstanding',  # 'выдающийся',
'funny',  # 'смешной',
'refreshing', # «освежающий»
'classic',  # классический
'unique',  # уникальный
'lovely',  # прекрасный
# 'good', # хороший - понижает оценку
# 'worthy',  # достойный - понижает оценку
'modern'  # современный
```

```
)

negative_words = (
'1/10',
'2/10',
'3/10',
'4/10',
'worst', # 'худший',
'awful',  # 'ужасный',
'crap',  # 'дерьмо',
'shit',  # 'дерьмо',
'sucks',  # 'отстой',
'boring',  # 'скучный',
'trash',  # 'мусор',
'mediocre',  # 'посредственный',
'mess',  # 'беспорядок',
'bad',  # 'Плохо',
'awful',  # 'ужасный',
'worthless',  # «бесполезный»,
'stupid',  # 'глупый',
'banal',  # 'банальный',
'boring',  # 'скучный',
'fail',  # 'потерпеть неудачу',
'disgusting',  # 'отвратительный',
'hate',  # 'ненавидеть'
'crappy',  # дрянной
# 'poor'  # бедный - понижает оценку
# 'sinister'  # зловещий - понижает оценку
)

positives_count = test_df.review.apply(lambda text: sum(word in text for word in positive_words))
negatives_count = test_df.review.apply(lambda text: sum(word in text for word in negative_words))
is_positive = positives_count > negatives_count
correct_count = (is_positive == test_df.is_positive).values.sum()

accuracy = correct_count / len(test_df)

print('Test accuracy = {:.2%}'.format(accuracy))
if accuracy > 0.71:
    from IPython.display import Image, display
#     display(Image('https://s3.amazonaws.com/achgen360/t/rmmoZsub.png', width=500))
    display(Image('https://www.freeiconspng.com/uploads/success-icon-1.png', width=500))
```

```
Test accuracy = 74.55%
```

B [4]:

Out[4]:
```
22    9
32    4
33    4
60    5
66    5
Name: review, dtype: int64
```

In [9]:
```
ids = [22, 215, 1598, 2579, 3114]
```

In [6]:

Out[6]:
```
4     11
11     4
28     5
74     4
75     5
Name: review, dtype: int64
```

In [7]:
```
ids = [4, 115, 597, 760]
```

Задание Придумайте хорошие ключевые слова или фразы и наберите хотя бы 71% точности на тесте (и не забудьте посмотреть на код классификации!)

Задание Кому-нибудь нравятся эти `<br /><br />` ? Лично мне - нет. Напишите регулярку, которая будет их удалять

In [10]:
```
import re

pattern = re.compile('<br />')

print(train_df['review'].iloc[3])
```

```
Spoilers ahead if you want to call them that...<br /><br />I would almost recommend this film just so people can trul
y see a 1/10. Where to begin, we'll start from the top...<br /><br />THE STORY: Don't believe the premise - the movie
has nothing to do with abandoned cars, and people finially understanding what the mysterious happenings are. It's a d
raub, basic, go to cabin movie with no intensity or "effort".<br /><br />THE SCREENPLAY: I usually give credit to ind
ie screenwriters, it's hard work when you are starting out...but this is crap. The story is flat - it leaves you emot
ionless the entire movie. The dialogue is extremely weak and predictable boasting lines of "Woah, you totally freaked
me out" and "I was wondering if you'd uh...if you'd like to..uh, would you come to the cabin with me?". It makes me w
ant to rip out all my hair, one strand at a time and feed it to myself.<br /><br />THE CHARACTERS: HOLY CRAP!!!! Some
have described the characters as flat, I want to take it one step further and say that they actually have a reverse c
haracter arch.. They actually start working on a parallel universe and almost start acting backwards...<br /><br />TH
E ACTORS: Worse than the characters are the actors. They take already poor written characters and add in terrible hig
h school drama acting. The "Woah you totally freaked me out" was said so monotone and slow - like it was dumbed down.
I could complain for hours on the actors alone.<br /><br />TECHNICAL: LIGHTING: An eight year old would be disappoint
ed with lighting on this movie. Too shadowy in areas, too bleached in others. The director shouldn't use light as an
emotion until he learns how to light a basic scene properly. Baby Steps! SOUND: How many sound guys does it take to m
ake a really shotty sounding movie? 9. With that many sound guys this should sound amazing but quite the opposite has
occured. There is one scene in particular that really sticks out, these guys are driving in a car and the sound of th
e car changes with every camera angle....WEAK! CAMERA: Learn to use it.<br /><br />Anyway, I'm running out of complai
ning space.....rent it - I dare you...Rent it and learn from it...give it a 1 rating..it deserves it.<br /><br />Sign
ing off... Amanda Christmas
Spoilers ahead if you want to call them that...  I would almost recommend this film just so people can truly see a 1/
10. Where to begin, we'll start from the top...  THE STORY: Don't believe the premise - the movie has nothing to do w
ith abandoned cars, and people finially understanding what the mysterious happenings are. It's a draub, basic, go to
cabin movie with no intensity or "effort".  THE SCREENPLAY: I usually give credit to indie screenwriters, it's hard w
ork when you are starting out...but this is crap. The story is flat - it leaves you emotionless the entire movie. The
dialogue is extremely weak and predictable boasting lines of "Woah, you totally freaked me out" and "I was wondering
if you'd uh...if you'd like to..uh, would you come to the cabin with me?". It makes me want to rip out all my hair, o
ne strand at a time and feed it to myself.  THE CHARACTERS: HOLY CRAP!!!! Some have described the characters as flat,
I want to take it one step further and say that they actually have a reverse character arch.. They actually start wor
king on a parallel universe and almost start acting backwards...  THE ACTORS: Worse than the characters are the actor
s. They take already poor written characters and add in terrible high school drama acting. The "Woah you totally frea
ked me out" was said so monotone and slow - like it was dumbed down. I could complain for hours on the actors alone.
TECHNICAL: LIGHTING: An eight year old would be disappointed with lighting on this movie. Too shadowy in areas, too b
leached in others. The director shouldn't use light as an emotion until he learns how to light a basic scene properl
y. Baby Steps! SOUND: How many sound guys does it take to make a really shotty sounding movie? 9. With that many soun
d guys this should sound amazing but quite the opposite has occured. There is one scene in particular that really sti
cks out, these guys are driving in a car and the sound of the car changes with every camera angle....WEAK! CAMERA: Le
arn to use it.  Anyway, I'm running out of complaining space.....rent it - I dare you...Rent it and learn from it...g
ive it a 1 rating..it deserves it.  Signing off... Amanda Christmas
```

Применим ее:

In [11]:
```
train_df['review'] = train_df['review'].apply(lambda text: pattern.subn(' ', text)[0])
```

In [12]:
```
def replace_patern(text, patern, replace_letter=' '):
    """
    Заменим патерн на пробелы.
    """
    text = re.sub(patern, replace_letter, text)
```

In [13]:

Out[13]:

```
is_positive
0
review         This show comes up with interesting locations as fast as the travel channel. It is billed as reality b
ut in actuality it is pure prime time soap opera. It's tries to use exotic locales as a facade to bring people into a
phony contest & then proceeds to hook viewers on the contestants soap opera style.  It also borrows from an early CBS
game show pioneer- Beat The Clock- by inventing situations for its contestants to try & overcome. Then it rewards the
winner money. If they can spice it up with a little interaction between the characters, even better. While the game f
ormat is in slow motion versus Beat The Clock- the real accomplishment of this series is to escape reality.   This sh
ow has elements of several types of successful past programs. Reality television, hardly, but if your hooked on the c
ontestants, locale or contest, this is your cup of tea. If your not, this entire series is as I say, drivel dripping
with gravy. It is another show hiding behind the reality label which is the trend it started in 2000.  It is slick &
```

B [14]:
```
train_df['review'] = train_df['review'].apply(replace_patern, patern = pattern)
test_df['review'] = test_df['review'].apply(replace_patern, patern = pattern)
```

Out[14]:
```
is_positive
0
review         This show comes up with interesting locations as fast as the travel channel. It is billed as reality b
ut in actuality it is pure prime time soap opera. It's tries to use exotic locales as a facade to bring people into a
phony contest & then proceeds to hook viewers on the contestants soap opera style.  It also borrows from an early CBS
game show pioneer- Beat The Clock- by inventing situations for its contestants to try & overcome. Then it rewards the
winner money. If they can spice it up with a little interaction between the characters, even better. While the game f
ormat is in slow motion versus Beat The Clock- the real accomplishment of this series is to escape reality.   This sh
ow has elements of several types of successful past programs. Reality television, hardly, but if your hooked on the c
ontestants, locale or contest, this is your cup of tea. If your not, this entire series is as I say, drivel dripping
with gravy. It is another show hiding behind the reality label which is the trend it started in 2000.  It is slick &
well produced, so it might last a while yet. After all, so do re-runs of Gilligan's Island, Green Acres, The Beverly
Hillbillies & The Brady Bunch. This just doesn't employ professional actors. The intelligence level is about the sam
e.
Name: 1, dtype: object
```

B [15]:
```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(token_pattern=r'(?u)\b\w\w+\b')

dummy_data = ['The movie was excellent',
              'the movie was awful']

dummy_matrix = vectorizer.fit_transform(dummy_data)

print(dummy_matrix.toarray())
```

```
[[0 1 1 1 1]
 [1 0 1 1 1]]
['awful', 'excellent', 'movie', 'the', 'was']

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

B [16]:
```
vectorizer = CountVectorizer()
```

Out[16]:
```
CountVectorizer()
```

B [17]:
```
# Находим мусорные слова и включаем их в стоп-лист
# print(dummy_matrix.toarray())
# print(vectorizer.get_feature_names())
# vectorizer.get_feature_names()[1145:1300]
stop_words = vectorizer.get_feature_names()[:1145]  # мусорные слова
```

B [18]:

Out[18]:
```
['00',
 '000',
 '0000000000001',
 '00001',
 '00015',
 '000s',
 '001',
 '003830',
 '006',
 '007',
 '0079',
 '0080',
 '0083',
 '0093638',
 '00am',
 '00pm',
 '00s',
 '01',
 '01pm',
```

B [19]:

Out[19]:
```
<1x74849 sparse matrix of type '<class 'numpy.int64'>'
        with 206 stored elements in Compressed Sparse Row format>
```

B [20]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

dummy_data = ['The movie was excellent',
              'the movie was awful']
dummy_labels = [1, 0]

vectorizer = CountVectorizer()
classifier = LogisticRegression()

model = Pipeline([
    ('vectorizer', vectorizer),
    ('classifier', classifier)
])

model.fit(dummy_data, dummy_labels)

print(vectorizer.get_feature_names())
```

```
['awful', 'excellent', 'movie', 'the', 'was']
[[-0.40104279  0.40104279  0.          0.          0.        ]]

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

B [21]:

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[21]:
```
Pipeline(steps=[('vectorizer', CountVectorizer()),
                ('classifier', LogisticRegression())])
```

B [22]:
```
from sklearn.metrics import accuracy_score

def eval_model(model, test_df):
    preds = model.predict(test_df['review'])
    print('Test accuracy = {:.2%}'.format(accuracy_score(test_df['is_positive'], preds)))
```

```
Test accuracy = 86.54%
```

B [23]:
```
import eli5
```

```
C:\ProgramData\Anaconda3\lib\site-packages\xgboost\compat.py:31: FutureWarning: pandas.Int64Index is deprecated and w
ill be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

Out[23]:

y=1 top features

Weight[?]	Feature
+1.876	refreshing
+1.778	wonderfully
+1.708	funniest
+1.663	surprisingly
+1.640	rare
+1.436	superb
+1.396	excellent
+1.374	incredible
+1.337	perfect
+1.315	delightful
+1.279	vengeance
… 37687 more positive …	
… 37123 more negative …	
-1.273	worse
-1.275	uninteresting
-1.309	mildly
-1.313	redeeming

Weight[?]	Feature
-1.364	weak
-1.365	baldwin
-1.367	wooden
-1.382	badly
-1.389	horrible
-1.419	mst3k
-1.423	save
-1.429	lame
-1.464	mediocre
-1.551	pointless
-1.585	unfunny
-1.591	alright
-1.604	forgettable
-1.611	disappointing
-1.645	boring
-1.694	avoid
-1.709	awful
-1.731	fails
-1.741	laughable
-1.997	mess
-2.164	lacks
-2.304	worst
-2.365	poorly
-2.620	waste
-2.674	disappointment

B [25]:
```
print('Positive' if test_df['is_positive'].iloc[1] else 'Negative')
eli5.show_prediction(classifier, test_df['review'].iloc[1], vec=vectorizer,
```

```
Positive
```

Out[25]: **y=positive** (probability **1.000**, score **18.938**) top features

Contribution[?]	Feature
+18.954	Highlighted in text (sum)
-0.016	<BIAS>

this is both an entertaining and a touching version of the classic tale, also quite intelligent, not of the 'me tarzan, you jane' school at all. it's the famous story of a child reared to manhood in the jungle by apes. a titled british couple (the wife pregnant) is stranded in the african wilds after a shipwreck. after the parents' deaths, the baby is raised in the jungle by apes. twenty years later, this young man (i.e. tarzan) rescues a wounded belgian explorer, nursing him back to health. the belgian discovers evidence that his rescuer is the young lord greystoke and returns him to his rightful estate in scotland, where he must adjust to civilized society. the movie is sort of divided into two parts. in the first half, we see tarzan in his jungle environment. not being an expert, i am unaware as to the realism of its depiction of ape community life, but it is certainly entertaining. for me, the more moving section is the second half, when tarzan must meet his real family, develop language skills, and adjust to aristocratic british society, all the while wooing jane (andie macdowell). he is portrayed as a 'noble savage', whether in the wild or in elegant edwardian parlors. by contrast, the upper crust is depicted as often far more barbaric than the jungle tarzan left. christopher lambert is fantastic in his sympathetic portrayal of tarzan in both the jungle and civilized environments. he conveys a real sense of his confusion and conflict, torn as he is between the two very different worlds, his original ape family and his new human one. sir ralph richardson, one of the old british legends, is brilliant as always in the role of tarzan's grandfather, the sixth earl of greystoke. the film focuses more on tarzan's struggles in adapting to civilization and his inner conflict than on his jungle exploits. this unusual take on the old classic makes it both the typical dramatic adventure but also, above all, a moving personal story. i wasn't surprised to note here that its director is the same individual, hugh hudson, who also directed chariots of fire, another brilliant movie.

Примеры неправильной классификации:

B [27]:
```
import numpy as np

preds = model.predict(test_df['review'])
incorrect_pred_index = np.random.choice(np.where(preds != test_df['is_positive'])[0])

eli5.show_prediction(classifier, test_df['review'].iloc[incorrect_pred_index],
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in stead.
  warnings.warn(msg, category=FutureWarning)
```

Out[27]: **y=positive** (probability **0.964**, score **3.281**) top features

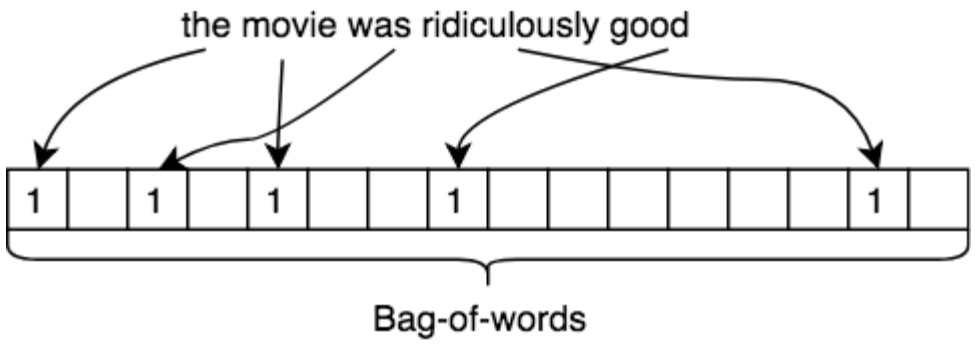
Contribution[?]	Feature
+3.296	Highlighted in text (sum)
-0.016	<BIAS>

james stewart plays johnny mason, lawyer. carole lombard is jane mason, wife. lucile watson the mother-in-law harriet mason. johnny sees jane and quickly marries her. mother is disappointed. mother lives with them. many troubles are ahead. jane can't cook. can't set the table. can't do many things according to mother. the interaction between daughter-in-law and mother are the highlights of this film. stewart and lombard are married but just don't have any real magic on screen. stewart is stewart. he is good as a timid husband and son but this doesn't carry the film. can baby mason build bridges between jane and harriet? a believable film for those that are married.

Пора переходить к машинке!

Как будем представлять текст? Проще всего - мешком слов.

Заведём большой-большой словарь - список всех слов в обучающей выборке. Тогда каждое предложение можно представить в виде вектора, в котором будет записано, сколько раз встретилось каждое из возможных слов:



Простой и приятный способ сделать это - запихнуть тексты в `CountVectorizer` .

Он имеет такую сигнатуру:

```
CountVectorizer(input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True,
preprocessor=None, tokenizer=None, stop_words=None, token_pattern=r'(?u)\b\w\w+\b', ngram_range=(1, 1), analy
zer='word', max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.int64
'>)
```

Для начала обратим внимание на параметры `lowercase=True` и `max_df=1.0, min_df=1, max_features=None` - они про то, что по умолчанию все слова будут приводиться к нижнему регистру и в словарь попадут все слова, встречавшиеся в текстах.

При желании можно было бы убрать слишком редкие или слишком частотные слова - пока не будем этого делать.

Посмотрим на простом примере, как он будет работать:

В [28]:
```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()

dummy_data = ['The movie was excellent',
              'the movie was awful']

dummy_matrix = vectorizer.fit_transform(dummy_data)

print(dummy_matrix.toarray())
```

```
[[0 1 1 1 1]
 [1 0 1 1 1]]
['awful', 'excellent', 'movie', 'the', 'was']
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

Как именно vectorizer определяет границы слов? Обратите внимание на параметр `token_pattern=r'(?u)\b\w\w+\b'` - как он будет работать?

Запустим его на реальных данных:

В [29]:
```
vectorizer = CountVectorizer()
```

Out[29]:
```
CountVectorizer()
```

Посмотрим на слова, попавшие в словарь:

В [30]:

Out[30]:

```
Г'00'.
```

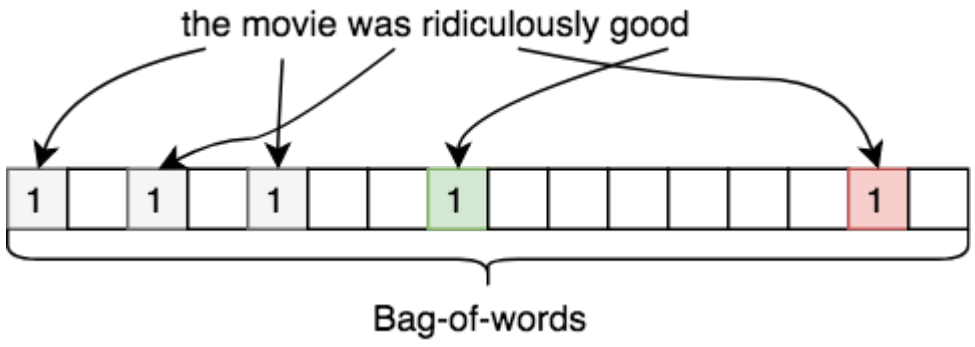
Попробуем кого-нибудь таки сконвертировать

В [31]:

Out[31]:
```
<1x74849 sparse matrix of type '<class 'numpy.int64'>'
        with 206 stored elements in Compressed Sparse Row format>
```

То, что и хотели - вектор с bow (т.е. bag-of-words) представлением исходного текста.

И чем эта информация может помочь? Ну, всё тем же - какие-то слова носят положительный окрас, какие-то - отрицательный. Большинство вообще нейтральный, да.



Хочется, наверное, подобрать коэффициенты, которые будут определять уровень окраса, да? Подбирать нужно по обучающей выборке, а не как мы перед этим делали.

Например, для выборки

```
1   The movie was excellent
0   the movie was awful
```

легко подобрать коэффициенты на глазок: что-нибудь вроде `+1` для `excellent`, `-1` для `awful` и по нулям всем остальным.

Построим линейную модель, которая станет этим заниматься. Она будет учиться строить разделяющую гиперплоскость в пространстве bow-векторов.

Проверим, как справится логистическая регрессия с нашей супер-выборкой из пары предложений

В [32]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

dummy_data = ['The movie was excellent',
              'the movie was awful']
dummy_labels = [1, 0]

vectorizer = CountVectorizer()
classifier = LogisticRegression()

model = Pipeline([
    ('vectorizer', vectorizer),
    ('classifier', classifier)
])

model.fit(dummy_data, dummy_labels)
```

```
print(vectorizer.get_feature_names())
```

```
['awful', 'excellent', 'movie', 'the', 'was']
[[-0.40104279  0.40104279  0.          0.          0.        ]]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

Получилось что надо.

Запустим теперь её на реальных данных.

В [33]:

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
```

Out[33]:
```
Pipeline(steps=[('vectorizer', CountVectorizer()),
                ('classifier', LogisticRegression())])
```

B [14]:
```
from sklearn.metrics import accuracy_score

def eval_model(model, test_df):
    preds = model.predict(test_df['review'])
    print('Test accuracy = {:.2%}'.format(accuracy_score(test_df['is_positive'], preds)))
```

```
Test accuracy = 86.50%
```

1. Посмотрите на токены если будут мусорные добавьте их в стоп слова и обучите заново

B [38]:

Out[38]:
```
['00',
 '000',
 '0000000000001',
 '00001',
 '00015',
 '000s',
 '001',
 '003830',
 '006',
 '007']
```

B [40]:
```
vectorizer = CountVectorizer(stop_words=stop_words)
classifier = LogisticRegression()

model = Pipeline([
    ('vectorizer', vectorizer),
    ('classifier', classifier)
])

model.fit(train_df['review'], train_df['is_positive'])
# print(vectorizer.get_feature_names())
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[40]:
```
Pipeline(steps=[('vectorizer',
                 CountVectorizer(stop_words=['00', '000', '0000000000001',
                                             '00001', '00015', '000s', '001',
                                             '003830', '006', '007', '0079',
                                             '0080', '0083', '0093638', '00am',
                                             '00pm', '00s', '01', '01pm', '02',
                                             '020410', '029', '03', '04', '041',
                                             '05', '050', '06', '06th', '07', ...])),
                ('classifier', LogisticRegression())])
```

B [42]:

```
Test accuracy = 86.81%
```

Без использования stop_words:
Test accuracy = 86.50%

С использованием stop_words:
Test accuracy = 86.81%

Прогресс!

Хочется как-то посмотреть, что заинтересовало классификатор. К счастью, сделать это совсем просто:

B [15]:
```
import eli5
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is
deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out inste
ad.
  warnings.warn(msg, category=FutureWarning)
```

Out[15]: **y=1** top features

Weight[?]	Feature
+1.855	refreshing
+1.760	wonderfully
+1.689	funniest
+1.647	surprisingly
+1.626	rare
+1.432	superb
+1.401	excellent
+1.365	incredible
+1.351	perfect
+1.302	delightful
+1.264	vengeance
… 37681 more positive …	
… 37129 more negative …	
-1.280	dull
-1.281	worse
-1.295	mildly
-1.303	redeeming
-1.349	baldwin
-1.355	wooden
-1.357	weak
-1.379	badly
-1.393	horrible
-1.405	mst3k
-1.416	save
-1.423	lame
-1.452	mediocre
-1.541	pointless
-1.569	alright
-1.570	unfunny
-1.588	forgettable
-1.605	disappointing
-1.653	boring
-1.686	avoid
-1.723	fails
-1.725	awful
-1.727	laughable
-1.983	mess
-2.143	lacks
-2.320	worst
-2.350	poorly
-2.615	waste
-2.648	disappointment

Посмотрим на конкретные примеры его работы:

B [43]:
```
print('Positive' if test_df['is_positive'].iloc[1] else 'Negative')
eli5.show_prediction(classifier, test_df['review'].iloc[1], vec=vectorizer,
```

```
Positive

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

Out[43]: **y=positive** (probability **1.000**, score **15.827**) top features

Contribution[?]	Feature
+15.874	Highlighted in text (sum)
-0.047	<BIAS>

this is both an entertaining and a touching version of the classic tale, also quite intelligent, not of the 'me tarzan, you jane' school at all. it's the famous story of a child reared to manhood in the jungle by apes. a titled british couple (the wife pregnant) is stranded in the african wilds after a shipwreck. after the parents' deaths, the baby is raised in the jungle by apes. twenty years later, this young man (i.e. tarzan) rescues a wounded belgian explorer, nursing him back to health. the belgian discovers evidence that his rescuer is the young lord greystoke and returns him to his rightful estate in scotland, where he must adjust to civilized society. the movie is sort of divided into two parts. in the first half, we see tarzan in his jungle environment. not being an expert, i am unaware as to the realism of its depiction of ape community life, but it is certainly entertaining. for me, the more moving section is the second half, when tarzan must meet his real family, develop language skills, and adjust to aristocratic british society, all the while wooing jane (andie macdowell). he is portrayed as a 'noble savage', whether in the wild or in elegant edwardian parlors. by contrast, the upper crust is depicted as often far more barbaric than the jungle tarzan left. christopher lambert is fantastic in his sympathetic portrayal of tarzan in both the jungle and civilized environments. he conveys a real sense of his confusion and conflict, torn as he is between the two very different worlds, his original ape family and his new human one. sir ralph richardson, one of the old british legends, is brilliant as always in the role of tarzan's grandfather, the sixth earl of greystoke. the film focuses more on tarzan's struggles in adapting to civilization and his inner conflict than on his jungle exploits. this unusual take on the old classic makes it both the typical dramatic adventure but also, above all, a moving personal story. i wasn't surprised to note here that its director is the same individual, hugh hudson, who also directed chariots of fire, another brilliant movie.

B [44]:
```
print('Positive' if test_df['is_positive'].iloc[6] else 'Negative')
eli5.show_prediction(classifier, test_df['review'].iloc[6], vec=vectorizer,
```

```
Negative

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

Out[44]:

y=positive (probability **0.000**, score **-8.378**) top features

Contribution[?]	Feature
-0.047	<BIAS>
-8.332	Highlighted in text (sum)

this movie is terrible, it was so difficult to believe that katie became a heartfelt teenager with the power to save the pity chinese people, the movie didn't show any convincing argument to prove that. and the rest of the plot didn't make any effort to show us more than a cheap common sense... the plot is ridiculous and the only thing we can extract from it is that it demonstrate how arrogant a human can be. katie must have inherited her arrogance from her mother, the most annoying character i have seen for a long time. the acting and scenery were ok, but the plot ruins everything, full of cheap clichés and hypocritical scenes, i expect not to see this movie again in my life. skip this one!

Посмотрим на примеры неправильной классификации, наконец:

B [45]:
```
import numpy as np

preds = model.predict(test_df['review'])
incorrect_pred_index = np.random.choice(np.where(preds != test_df['is_positive'])[0])

eli5.show_prediction(classifier, test_df['review'].iloc[incorrect_pred_index],
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

Out[45]:

y=positive (probability **0.006**, score **-5.066**) top features

Contribution[?]	Feature
-0.047	<BIAS>
-5.020	Highlighted in text (sum)

black comedy isn't always an easy sell. every now and then you get a black comedy that is hugely successful, like fargo, for example. but usually they don't often find big audiences. people seem to either set their minds for comedy, or for serious mayhem. there doesn't seem to be a big market for a good mixture of both. throw momma from the train was a fairly decent hit, yet few people seem to remember much about it in this day and age. danny devito just about hit this one all the way out of the park back in 1987. devito plays an odd mamma's boy named owen looking to rid himself of his outrageously overbearing and unpleasant mother whom he still lives with. the mother is played by anne ramsey, who passed away shortly after this was released, and she is quite a caricature. she is loud, ugly, rude, and overbearing. though owen hardly seems like he could take care of himself, he wants desperately to have his mother offed. he fantasizes about it in some truly weird scenes, but he clearly doesn't have the guts to actually do it himself. that's where billy crystal comes in. crystal plays larry donner, owen's creative writing teacher at a nearby community college. larry is a paranoid would-be intellectual novelist who claims his ex-wife stole his novel and made millions off it. he is currently trying to write a new one, but cannot even come up with a decent first sentence. "the night was...." owen hears larry wish his ex-wife were dead during an outburst at the school cafeteria. and borrowing the idea from strangers on a train, owen decides to travel to hawai'i and murder larry's ex-wife. once it appears he has done so, he expects larry to return the favor and kill his mother. the resulting action is often quite funny, and even poignant. it's certainly never dull and often full of surprises. the acting is exceptional, even if ramsey was a bit over the top. crystal is as good as he can be, and devito has always been undervalued as a performer. the film relies on quite a bit of physical comedy which usually works, often painfully so. the film makes use of some truly innovative editing techniques in some scenes, and the off-beat tone is truly refreshing. i have often been critical of the late 1980s as being a time of artistic malaise and down right lazy film-making. throw momma from the train takes chances. both in how its characters are drawn as well as its general plot. how many comedies revolve around a son having his mother murdered? the film isn't too long, and it is chock full of laughs. writers are apt to find it more interesting than the general public, but it can still be enjoyed by just about anyone. 9 of 10 stars. the hound.

Придумываем новые признаки

Tf-idf

Сейчас мы на все слова смотрим с одинаковым весом - хотя какие-то из них более редкие, какие-то более частые, и эта частотность - полезная, вообще говоря, информация.

Самый простой способ добавить статистическую информацию о частотностях - сделать *tf-idf* взвешивание:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t)$$

tf - term-frequency - частотность слова `t` в конкретном документе `d` (рецензии в нашем случае). Это ровно то, что мы уже считали.

idf - inverse document-frequency - коэффициент, который тем больше, чем в меньшем числе документов встречалось данное слово. Считается как-нибудь так:

$$\text{idf}(t) = \log \frac{1 + n_d}{1 + n_{d(t)}} + 1$$

где n_d - число всех документов, а $n_{d(t)}$ - число документов со словом `t`.

Использовать его просто - нужно заменить `CountVectorizer` на `TfidfVectorizer`.

Задание Попробуйте запустить `TfidfVectorizer`. Посмотрите на ошибки, которые он научился исправлять, и на ошибки, которые он начал делать - по сравнению с `CountVectorizer`.

В [50]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words=stop_words)
classifier = LogisticRegression()

model = Pipeline([
    ('vectorizer', vectorizer),
    ('classifier', classifier)
])

model.fit(train_df['review'], train_df['is_positive'])
```

```
Test accuracy = 88.29%
```

N-граммы слов

До сих пор мы смотрели на тексты как на мешок слов - но очевидно, что есть разница между `good movie` и `not good movie`.

Добавим информацию (хоть какую-то) о последовательностях слов - будем извлекать еще и биграммы слов.

В Vectorizer'ах для этого есть параметр `ngram_range=(n_1, n_2)` - он говорит, что нужны n_1-...n_2-граммы.

Задание Попробуйте увеличенный range и поинтерпретируйте полученный результат.

В [55]:
```
vectorizer = TfidfVectorizer(ngram_range=(1, 2), stop_words=stop_words, min_df = 0.002, max_df = 0.95)
classifier = LogisticRegression()

model = Pipeline([
    ('vectorizer', vectorizer),
    ('classifier', classifier)
])

model.fit(train_df['review'], train_df['is_positive'])
```

```
Test accuracy = 89.65%
```

Удалось добиться увеличения accuracy с 88.64% до 89.65%

N-граммы символов

Символьные n-граммы дают простой способ выучить полезные корни и суффиксы, не связываясь с этой вашей лингвистикой - только статистика, только хардкор.

Например, слово `badass` мы можем представить в виде такой последовательности триграмм:

```
##b #ba bad ada das ass ss# s##
```

So interpretable, неправда ли?

Реализовать это дело всё так же просто - нужно поставить `analyzer='char'` в вашем любимом Vectorizer'е и выбрать размер `ngram_range`.

Задание Запилите классификатор на n-граммах символов и визуализируйте его.

В [56]:
```
vectorizer = TfidfVectorizer(ngram_range=(2, 6), max_features=20000, analyzer='char', stop_words=stop_words)
classifier = LogisticRegression()

model = Pipeline([
    ('vectorizer', vectorizer),
    ('classifier', classifier)
])
```

```
model.fit(train_df['review'], train_df['is_positive'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(

Test accuracy = 87.86%
```

B [57]:
```
print('Positive' if test_df['is_positive'].iloc[1] else 'Negative')
eli5.show_prediction(classifier, test_df['review'].iloc[1], vec=vectorizer,
```

```
Positive

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names
is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

Out[57]: **y=positive** (probability **0.962**, score **3.245**) top features

Contribution?	Feature
+3.194	Highlighted in text (sum)
+0.050	<BIAS>

this is both an entertaining and a touching version of the classic tale, also quite intelligent, not of the 'me tarzan, you jane' school at all. it's the famous story of a child reared to manhood in the jungle by apes. a titled british couple (the wife pregnant) is stranded in the african wilds after a shipwreck. after the parents' deaths, the baby is raised in the jungle by apes. twenty years later, this young man (i.e. tarzan) rescues a wounded belgian explorer, nursing him back to health. the belgian discovers evidence that his rescuer is the young lord greystoke and returns him to his rightful estate in scotland, where he must adjust to civilized society. the movie is sort of divided into two parts. in the first half, we see tarzan in his jungle environment. not being an expert, i am unaware as to the realism of its depiction of ape community life, but it is certainly entertaining. for me, the more moving section is the second half, when tarzan must meet his real family, develop language skills, and adjust to aristocratic british society, all the while wooing jane (andie macdowell). he is portrayed as a 'noble savage', whether in the wild or in elegant edwardian parlors. by contrast, the upper crust is depicted as often far more barbaric than the jungle tarzan left. christopher lambert is fantastic in his sympathetic portrayal of tarzan in both the jungle and civilized environments. he conveys a real sense of his confusion and conflict, torn as he is between the two very different worlds, his original ape family and his new human one. sir ralph richardson, one of the old british legends, is brilliant as always in the role of tarzan's grandfather, the sixth earl of greystoke. the film focuses more on tarzan's struggles in adapting to civilization and his inner conflict than on his jungle exploits. this unusual take on the old classic makes it both the typical dramatic adventure but also, above all, a moving personal story. i wasn't surprised to note here that its director is the same individual, hugh hudson, who also directed chariots of fire, another brilliant movie.

Подключаем лингвистику

Лемматизация и стемминг

Если присмотреться, можно найти формы одного слова с разной семантической окраской по мнению классификатора. Или нет?

Задание Найти формы слова с разной семантической окраской.

Поверя, что они есть, попробуем что-нибудь с этим сделать.

Например, лемматизируем - сведем к начальной форме все слова. Поможет в этом библиотека spacy.

B [71]:
```
# !python -m spacy download en
# !python -m spacy link en_core_web_sm en
# !pip install https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.3.1/en_core_web_sm-2.3.1.t
```

B [69]:
```
import spacy
from spacy import displacy

nlp = spacy.load('en_core_web_sm', disable=['parser'])
# nlp = spacy.load('en', disable=['parser'])
```

B [70]:
```
for token in docs[0]:
```

```
Dreamgirls dreamgirl O
, , O
despite despite O
its its O
fistful fistful O
of of O
Tony Tony B PERSON
wins win O
in in O
an an B DATE
incredibly incredibly I DATE
weak weak I DATE
year year I DATE
on on O
```

2. Проверьте изменилось ли качество при лемматизации/и без неё

B [82]:
```
# стоп-слова
from spacy.lang.en.stop_words import STOP_WORDS
stopwords = list(STOP_WORDS) + stop_words
```

```
['for', '’ve', 'eleven', 'third', 'then', 'name', 'namely', 'though', 'be', 'while', "'d", 'even', 'either', 'myself
', 'when', 'there', 'since', 'himself', 'we', 'herein', 'where', 'my', 'this', 'being', 'against', 'could', 'yet', 'b
ut', 'own', 'further', 'take', 'make', 'anything', 'with', 'had', 'serious', 'somehow', 'seeming', 'call', 'well', 'i
', 'same', 'it', 'whereby', 'various', 'meanwhile', 'whoever', 'alone', 'until', 'always', 'hence', 'both', 'front',
'of', 'ten', 'anyone', '‘ll', 'wherever', 'why', 'only', 'cannot', 'go', '’d', 'forty', 'as', "'ll", 'twenty', 'out',
'which', 'except', '’ll', 'elsewhere', 'were', 'already', 'really', 'whence', "'m", 'seem', 'his', 'six', 'whatever',
'other', 'none', 'that', 'thereby', 'without', 'one', 'done', 'rather', 'keep', 'he', 'above', 'still', 'enough', 'fi
fteen', 'over', 'who', 'some', 'sixty', 'here', 'move', 'among', 'again', 'anyway', 'much', 'or', 'these', 'twelve',
'its', 'most', 'between', 'part', 'hundred', 'someone', 'fifty', 'if', 'into', 'us', 'within', "'ve", 'hereafter', 't
herefore', 'every', 'mine', 'down', 'them', 'once', 'their', 'herself', 'say', 'do', 'due', 'is', 'seems', 'unless',
'not', '’s', 'full', 'afterwards', 'from', 'may', 'give', 'have', 'towards', 'thus', 'themselves', 'never', 'in', 'ab
out', "'s", 'thence', 'would', 'yourselves', 'others', 'least', 'whole', 'any', 'thereafter', 'eight', 'three', '’re
', '‘m', 'wherein', 'almost', 'everything', 'by', 'her', '’m', 'onto', 'yours', '‘s', 'everyone', 'so', 'whereas', 'n
or', 'am', 'she', 'too', 'get', 'neither', 'moreover', 'and', 'now', 'itself', 'although', 'ever', 'ours', 'under', '
bottom', 'ca', 'our', 'no', 'whereupon', 'whenever', 'mostly', 'indeed', 'therein', 'please', 'anyhow', 'they', 'toge
ther', 'during', 'such', 'was', 'becoming', 'him', 'n‘t', 'all', 'are', 'nevertheless', 'show', 'doing', 'formerly',
'been', "n't", 'become', 'something', 'very', 'to', 'across', 'put', 'first', 'many', 'former', 'whether', 'using', '
what', 'noone', 'toward', 'five', 'side', 'somewhere', 'otherwise', 'became', 'another', 'last', 'amongst', 'via', 'q
uite', 'two', 'because', 'after', 'whom', 'has', 'how', 'should', 'seemed', 'beforehand', 'few', 'behind', 'hereupon
```

B [76]:
```
# Python spacy.tokens.Token() Examples - https://www.programcreek.com/python/example/114357/spacy.tokens.Token

def print_token(token):
    '''
    Print the important of a token
    @token: Token
    '''
    print('[token]')
    print('id\t%d' % token.i)
    print('text\t%s' % token.text)
    print('ner\t%s' % token.ent_type_)
    print('lemma\t%s' % token.lemma_)
    print('is_punct\t%s' % token.is_punct)
    print('is_stop\t%s' % token.is_stop)
    print('pos\t%s'% token.pos_)
    print('tag\t%s'% token.tag_)

for token in docs[0]:
```

...

B [83]:
```
# пунктуация
import string
```

2. Проверьте изменилось ли качество при лемматизации/и без неё

B [92]:
```
# лемматизация и токенизация
def data_clearning(sentence):
    doc = nlp(sentence)

    tokens = []
    for token in doc:
        # print(token)
        if token.ent_type_:
            tokens.append(token.ent_type_)
```

```
        elif token.lemma_:
            tokens.append(token.lemma_)
        else:
            tokens.appand(token.text)


    # Исключаем стоп-слова
    cln_tokens = []
    for token in tokens:
        if token not in stopwords and token not in punctuation:
            cln_tokens.append(token)
```

B [93]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.svm import LinearSVC
```

B [94]:
```
def eval_model(model, test_df):
    preds = model.predict(test_df['review'])
```

B [95]:
```
tfidf = TfidfVectorizer(tokenizer = data_clearning, ngram_range=(1, 2))
classifier = LogisticRegression()

model = Pipeline([
    ('tfidf', tfidf),
    ('clf', classifier)
])

model.fit(train_df['review'], train_df['is_positive'])
```

```
Test accuracy = 86.61%
```

Вывод:
результат с использованием леммитазации хуже чем без её использования, как с доп. настройками параметров (**86.61** против **89,65**), так и без дополнительных настроек параметров (**88,28**)

B [ ]:

Задание Сделайте классификатор на лемматизированных текстах.

Более простой способ нормализации слов - использовать стемминг. Он немного тупой, не учитывает контекст, но иногда оказывается даже эффективнее лемматизации - а, главное, быстрее.

По сути это просто набор правил, как обрезать слово, чтобы получить основу (stem):

B [110]:
```
from nltk import PorterStemmer

stemmer = PorterStemmer()

print(stemmer.stem('become'))
print(stemmer.stem('becomes'))
```

```
becom
becam
```

Задание Попробуйте вместо лемм классифицировать основы.

B [107]:
```
from nltk.corpus import stopwords

nltk.download('stopwords')


print('remove stop words')
```

```
remove stop words

[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\sil\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

https://github.com/rasbt/python-machine-learning-book/blob/master/code/optional-py-scripts/ch08.py (https://github.com/rasbt/python-machine-learning-book/blob/master/code/optional-py-scripts/ch08.py)

B [108]:
```
stop = stopwords.words('english')

def tokenizer_porter(text):
    tokens = [stemmer.stem(word) for word in text.split()]

    return [w for w in tokens if not w in stop]

def preprocessor(text):
    text = re.sub('<[^>]*>', '', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text)
    text = re.sub('[\W]+', ' ', text.lower()) +\
        ' '.join(emoticons).replace('-', '')
```

B [111]:
```
tfidf = TfidfVectorizer(
    preprocessor=preprocessor,
    tokenizer=tokenizer_porter,
    ngram_range=(1, 2)
)

classifier = LogisticRegression()

model = Pipeline([
    ('tfidf', tfidf),
    ('clf', classifier)
])

model.fit(train_df['review'], train_df['is_positive'])
```

```
Test accuracy = 88.64%
```

Результат лучше чем при использовании лемматизации (**88.61** вместо **86.61**).

NER

В текстах рецензий очень много именованных сущностей. Вот, например:

B [115]:

Dreamgirls, despite its fistful of Tony **PERSON** wins in an incredibly weak year **DATE** on Broadway **FAC**, has never been what one would call a jewel in the crown of stage musicals. However, that is not to say that in the right cinematic hands it could not be fleshed out and polished into something worthwhile on-screen. Unfortunately, what transfers to the screen is basically a slavishly faithful version of the stage hit with all of its inherent weaknesses intact. First **ORDINAL**, the score has never been one of the strong points of this production and the film does not change that factor. There are lots of songs (perhaps too many?), but few of them are especially memorable. The closest any come to catchy tunes are the title song and One Night Only **WORK_OF_ART** - the much acclaimed And I Am Telling You That I Am Not Going is less a great song than it is a dramatic set piece for the character of Effie **ORG** (Jennifer Hudson **PERSON**). The film is slick and technically well-produced, but the story and characters are surprisingly thin and lacking in any resonance. There is some interest in the opening moments, watching Jamie Foxx's **PERSON** Svengali-like manager manipulate his acts to the top, but that takes a back seat in the latter portion of the film, when the story conveniently tries to cast him as a villain, despite his having been right from a business stand-point for a good majority of the film. Beyonce Knowles **PERSON** is lovely and sings her songs perfectly well, but is stuck with a character who is basically all surface glitz. Anika Noni Rose **PERSON** as the third **ORDINAL** member of the Dreamgirls **ORG** trio literally has nothing to do for the entire film. Eddie Murphy **PERSON** acquits himself well as a singer obviously based on James Brown **PERSON**, but the role is not especially meaty and ultimately has little impact. Foxx would seem ideal casting, but he seems oddly withdrawn and bored. The film's biggest selling point is surely former American **NORP** Idol contestant/ Oscar **WORK_OF_ART** winner Jennifer Hudson **PERSON** in the central role of Effie **ORG** White **PRODUCT**, the temperamental singer who gets booted from the group and makes a triumphant closing act return. For me, Effie **ORG** has always been a big problem in both the show and the movie. The film obviously wants you to feel sorry for her and rather ham-handedly takes her side, but I have never been sure that this character deserves that kind of devotion. From the start, Effie **ORG** conducts herself for the most part like an obnoxious, egotistical, self-centered diva **ORG**, who is more interested in what everyone else can do for her rather than having

much vested interest in the group of which she is a part. When she is booted from the group for her unprofessionalism and bad attitude, the charges are more than well-founded, but the stage show/film seem to think Effie **ORG** should be cut unlimited slack simply because she has a great voice. Even though the film tries to soften some of Effie **ORG** 's harder edges to make her more likable, the charges still stand. Her story becomes more manipulative by suggesting she should have our further sympathy because she is an unwed mother struggling to raise her daughter - using the implication that (much like the talent card) motherhood immediately makes any behavior excusable. Indeed the only big effort the film makes to show Effie **ORG** 's mothering is to tell us about it and then include a scene where she barks at her daughter in the unemployment office, insists that the girl has "no father" and then refuse to look for gainful employment to support them since singing is all she knows. In the hands of a skillful actress, the gaps could perhaps have been remedied with technique and charisma. Unfortunately, Hudson **PERSON** is not that actress. She sings well, but the dialog-driven moments do not come naturally to her nor do high emotional moments. Effie **ORG** 's signature moment (the aforementioned And I Am Telling You... number) is well-sung by Hudson **PERSON** , but emotionally flat in the acting department. Effie **ORG** is supposed to expressing her rage and desperation at her predicament, but Hudson **PERSON** comes off as a cabaret performer belting out a hot number. All in all, not quite the emotional highlight one expects. The latter portion of the film is basically a predictable melange of events that maneuver Foxx **PERSON** into Hudson **ORG** 's earlier position and allow her to strut back in and lord it over everyone. Foxx **ORG** 's criminal offenses in the film are undoubtedly par for the course of many struggling record producers, but the film's seeming implication that he has it coming because he helped usher in the disco era is rather ridiculous, not to mention pretentious and condescending, particularly coming from a film with all of the depth of a puddle. The end result is a faithful rendition of the stage hit, drained of emotion, energy or anything that can be described as dynamic.

Вообще говоря, почему вдруг какой-нибудь Депп должен нести семантическую окраску? Однако оказывается, что классификатор выучивает, что какие-то имена чаще в положительных рецензиях - или наоборот. Это похоже на переобучение - почему бы не попробовать вырезать сущности?

Задание Удалите из текстов какие-то из сущностей, пользуясь координатами из запикленных файлов. Описание сущностей можно посмотреть здесь (https://spacy.io/api/annotation#named-entities). Запустите классификатор.

3. Замените все токены которые принадлежат сущностям на их тег. Проверьте изменилось ли качество после этого

B [116]:
```
from spacy.lang.en.stop_words import STOP_WORDS
stopwords = list(STOP_WORDS)
```

```
['for', '’ve', 'eleven', 'third', 'then', 'name', 'namely', 'though', 'be', 'while', "'d", 'even', 'either', 'myself
', 'when', 'there', 'since', 'himself', 'we', 'herein', 'where', 'my', 'this', 'being', 'against', 'could', 'yet', 'b
ut', 'own', 'further', 'take', 'make', 'anything', 'with', 'had', 'serious', 'somehow', 'seeming', 'call', 'well', 'i
', 'same', 'it', 'whereby', 'various', 'meanwhile', 'whoever', 'alone', 'until', 'always', 'hence', 'both', 'front',
'of', 'ten', 'anyone', '‘ll', 'wherever', 'why', 'only', 'cannot', 'go', '’d', 'forty', 'as', "'ll", 'twenty', 'out',
'which', 'except', '’ll', 'elsewhere', 'were', 'already', 'really', 'whence', "'m", 'seem', 'his', 'six', 'whatever',
'other', 'none', 'that', 'thereby', 'without', 'one', 'done', 'rather', 'keep', 'he', 'above', 'still', 'enough', 'fi
fteen', 'over', 'who', 'some', 'sixty', 'here', 'move', 'among', 'again', 'anyway', 'much', 'or', 'these', 'twelve',
'its', 'most', 'between', 'part', 'hundred', 'someone', 'fifty', 'if', 'into', 'us', 'within', "'ve", 'hereafter', 't
herefore', 'every', 'mine', 'down', 'them', 'once', 'their', 'herself', 'say', 'do', 'due', 'is', 'seems', 'unless',
'not', '’s', 'full', 'afterwards', 'from', 'may', 'give', 'have', 'towards', 'thus', 'themselves', 'never', 'in', 'ab
out', "'s", 'thence', 'would', 'yourselves', 'others', 'least', 'whole', 'any', 'thereafter', 'eight', 'three', '’re
', '‘m', 'wherein', 'almost', 'everything', 'by', 'her', '’m', 'onto', 'yours', '‘s', 'everyone', 'so', 'whereas', 'n
or', 'am', 'she', 'too', 'get', 'neither', 'moreover', 'and', 'now', 'itself', 'although', 'ever', 'ours', 'under', '
bottom', 'ca', 'our', 'no', 'whereupon', 'whenever', 'mostly', 'indeed', 'therein', 'please', 'anyhow', 'they', 'toge
ther', 'during', 'such', 'was', 'becoming', 'him', 'n‘t', 'all', 'are', 'nevertheless', 'show', 'doing', 'formerly',
'been', "n't", 'become', 'something', 'very', 'to', 'across', 'put', 'first', 'many', 'former', 'whether', 'using', '
what', 'noone', 'toward', 'five', 'side', 'somewhere', 'otherwise', 'became', 'another', 'last', 'amongst', 'via', 'q
uite', 'two', 'because', 'after', 'whom', 'has', 'how', 'should', 'seemed', 'beforehand', 'few', 'behind', 'hereupon
', 'nobody', 'top', 'besides', 'beyond', 'me', 'nine', 'throughout', 'also', 'else', 'nowhere', 'hereby', 'less', 'on
', 'more', 'off', 'perhaps', 'amount', 'the', 'everywhere', 'see', 'back', 'yourself', '‘d', 'often', 'must', 'beside
', 'empty', 'thereupon', 'your', 'an', 'a', 'along', 'below', 'just', 'will', '‘re', 'becomes', 'n’t', 'at', 'before
', 'each', 'thru', 'did', 'whither', 'latter', "'re", 'can', 'regarding', 'next', 'you', 'up', 'might', 'nothing', '‘
ve', 'ourselves', 'sometime', 'per', 'latterly', 'whereafter', 'four', 'those', 'sometimes', 'through', 'whose', 'upo
n', 'does', 'used', 'hers', 'anywhere', 'made', 'around', 're', 'however', 'several', 'than']
```

B [117]:
```
# токенизации, лемматизации, удаления именованных сущностей
def data_clearning_ner(sentence):
    doc = nlp(sentence)
```

```
    tokens = []
    for token in doc:
        if token.ent_type_ not in ['GPE', 'PERSON', 'ORG', 'TIME', 'MONEY']:
            # print(token)
            if token.ent_type_:
                tokens.append(token.ent_type_)
            elif token.lemma_:
                tokens.append(token.lemma_)
            else:
                tokens.append(token.text)


    # Исключаем стоп-слова
    cln_tokens = []
    for token in tokens:
        if token not in stopwords and token not in punctuation:
            cln_tokens.append(token)
```

B [118]:
```
tfidf = TfidfVectorizer(
    tokenizer = data_clearning_ner,
    ngram_range=(1, 2)
)

classifier = LogisticRegression()

model = Pipeline([
    ('tfidf', tfidf),
    ('clf', classifier)
])
model.fit(train_df['review'], train_df['is_positive'])
```

```
Test accuracy = 86.90%
```

Включаем дип лёрнинг

Мы тут пришли deep learning'ом заниматься, а делаем почему-то модель на логистической регрессии. Как так?

Попробуем запустить относительно стандартную модель для классификации текстов - сверточная сеть поверх словных эмбеддингов.

Разбираться, что это за зверь, будем на следующих занятиях, а пока будем просто им пользоваться :)

Каждое предложение нужно представлять набором слов - и сразу же начинаются проблемы. Во-первых, как ограничить длину предложения?

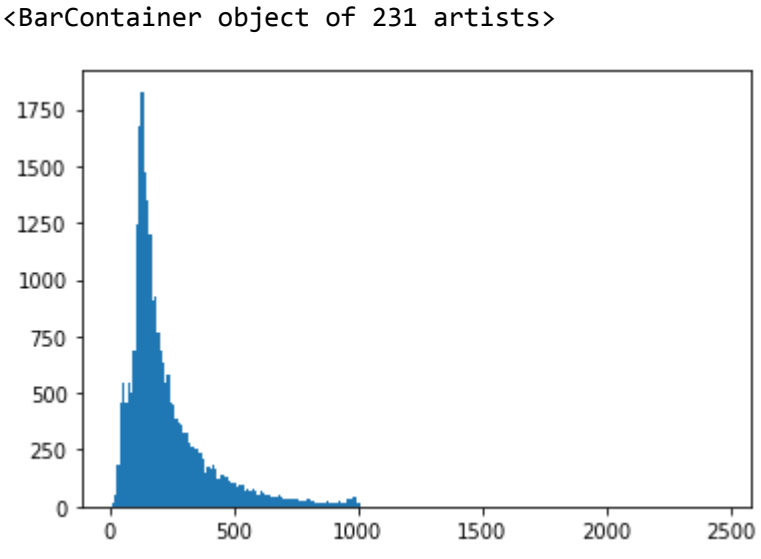
Прикинем по гистограмме, какая длина нам подходит:

B [ ]:

B [120]:
```
%matplotlib inline
import matplotlib.pyplot as plt

_, _, hist = plt.hist(train_df.review.apply(lambda text: len(text.split())), bins='auto')
```

Out[120]:
```
<BarContainer object of 231 artists>
```



Кроме этого, нужно перенумеровать как-то слова.

B [121]:
```
from collections import Counter

words_counter = Counter((word for text in train_df.review for word in text.lower().split()))

word2idx = {
    '': 0,
```

```
    '<unk>': 1
}
for word, count in words_counter.most_common():
    if count < 10:
        break

    word2idx[word] = len(word2idx)
```

```
Words count 26783
```

Задание Сконвертируйте данные

B [122]:
```
def convert(texts, word2idx, max_text_len):
    data = np.zeros((len(texts), max_text_len), dtype=np.int)

    for inx, text in enumerate(texts):
        result = []
        for word in text.split():
            if word in word2idx:
                result.append(word2idx[word])
        padding = [0]*(max_text_len - len(result))
        data[inx] = np.array(padding + result[-max_text_len:], dtype=np.int)
    return data

X_train = convert(train_df.review, word2idx, 1000)
```

```
<ipython-input-122-27b1b5540537>:2: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To sile
nce this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`,
you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, c
heck the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecat
ions (https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
  data = np.zeros((len(texts), max_text_len), dtype=np.int)
<ipython-input-122-27b1b5540537>:10: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To sil
ence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`,
you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, c
heck the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecat
ions (https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations)
  data[inx] = np.array(padding + result[-max_text_len:], dtype=np.int)
```

Поставим учиться модельку на keras.

Напоминание: на keras, чтобы обучить модель, нужно

1. Определить модель, например:

```
model = Sequential()
model.add(Dense(1, activation='sigmoid', input_dim=NUM_WORDS))
```

2. Задать функцию потерь и оптимизатор:

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

3. Запустить обучение:

```
model.fit(X_train, y_train,
          batch_size=32,
          epochs=3,
          validation_data=(X_test, y_test))
```

В NLP чаще всего ставятся задачи классификации, поэтому нужно запомнить такие функции потерь:

- **categorical_crossentropy** - для многоклассовой классификации, в качестве меток должны передаваться one-hot-encoding вектора
- **sparse_categorical_crossentropy** - аналогично предыдущему, но в качестве меток нужно передавать просто индексы соответствующих классов
- **binary_crossentropy** - для бинарной классификации

В качестве оптимизатора обычно используют `sgd` или `adam` .

B [123]:
```
import tensorflow as tf
```

Out[123]:
```
'2.9.0-dev20211226'
```

B [124]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
```

B [125]:
```
model = Sequential([
    Embedding(input_dim=len(word2idx), output_dim=64, input_shape=(X_train.shape[1],)),
    GlobalMaxPooling1D(),
    Dense(units=10, activation='relu'),
```

```
    Dense(units=10, activation='relu'),

    Dense(units=1, activation='sigmoid')
])

model.summary()
model.compile(loss='binary_crossentropy',
              optimizer='adam',
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 1000, 64)          1714112

 global_max_pooling1d (Globa  (None, 64)               0
 lMaxPooling1D)

 dense (Dense)               (None, 10)                650

 dense_1 (Dense)             (None, 10)                110

 dense_2 (Dense)             (None, 1)                 11

=================================================================
Total params: 1,714,883
Trainable params: 1,714,883
Non-trainable params: 0
_________________________________________________________________
```

B [126]:
```
model.fit(X_train, train_df.is_positive, batch_size=128, epochs=10,
```

```
Epoch 1/10
196/196 [==============================] - 39s 182ms/step - loss: 0.5911 - accuracy: 0.7214 - val_loss: 0.3914 - val_
accuracy: 0.8388
Epoch 2/10
196/196 [==============================] - 34s 173ms/step - loss: 0.2949 - accuracy: 0.8828 - val_loss: 0.2967 - val_
accuracy: 0.8709
Epoch 3/10
196/196 [==============================] - 34s 176ms/step - loss: 0.1806 - accuracy: 0.9345 - val_loss: 0.2948 - val_
accuracy: 0.8740
Epoch 4/10
196/196 [==============================] - 34s 172ms/step - loss: 0.1037 - accuracy: 0.9674 - val_loss: 0.3190 - val_
accuracy: 0.8710
Epoch 5/10
196/196 [==============================] - 32s 163ms/step - loss: 0.0526 - accuracy: 0.9871 - val_loss: 0.3510 - val_
accuracy: 0.8715
Epoch 6/10
196/196 [==============================] - 35s 177ms/step - loss: 0.0234 - accuracy: 0.9964 - val_loss: 0.3889 - val_
accuracy: 0.8692
Epoch 7/10
196/196 [==============================] - 34s 172ms/step - loss: 0.0099 - accuracy: 0.9992 - val_loss: 0.4239 - val_
accuracy: 0.8672
Epoch 8/10
196/196 [==============================] - 35s 177ms/step - loss: 0.0045 - accuracy: 0.9999 - val_loss: 0.4527 - val_
accuracy: 0.8672
Epoch 9/10
196/196 [==============================] - 34s 175ms/step - loss: 0.0028 - accuracy: 0.9999 - val_loss: 0.4738 - val_
accuracy: 0.8666
Epoch 10/10
196/196 [==============================] - 34s 174ms/step - loss: 0.0018 - accuracy: 0.9999 - val_loss: 0.4922 - val_
accuracy: 0.8661
```

Out[126]:
```
<keras.callbacks.History at 0x312ddacaf0>
```

Задание Подсчитайте качество модели на тесте

B [127]:
```
  loss, accuracy = model.evaluate(X_test, test_df.is_positive)
```

```
782/782 [==============================] - 11s 14ms/step - loss: 0.4922 - accuracy: 0.8661
Test loss 0.4922, accuracy 86.61%
```

B [ ]: