In [1]:
```
import os
import zipfile
import string
import numpy as np
import pandas as pd

from sklearn.preprocessing import MinMaxScaler, RobustScaler, Sta
ndardScaler
from sklearn.ensemble import RandomForestClassifier, StackingClas
sifier, VotingClassifier
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.feature_extraction.text import TfidfVectorizer, Coun
tVectorizer
from sklearn.metrics import log_loss, accuracy_score
from sklearn.model_selection import train_test_split, KFold, Grid
SearchCV, RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.decomposition import TruncatedSVD, NMF
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from nltk.corpus import stopwords
from nltk import word_tokenize, pos_tag
from nltk.stem.snowball import SnowballStemmer
from scipy.stats import uniform

import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

In [2]:
```
import nltk
# nltk.download('stopwords')
# nltk.download('punkt')
# nltk.download('averaged_perceptron_tagger')
```

In [3]:
```
sns.set(style='whitegrid')
```

Unzip data

In [4]:
```
INPUT_PATH = 'data/input'
IMAGE_PATH = 'img'
zip_file = 'spooky-author-identification.zip'
```

In [5]:
```
for path in [INPUT_PATH, IMAGE_PATH]:
    if not os.path.exists(path):
        os.makedirs(path)
```

In [6]:
```
os.listdir(INPUT_PATH)
```

Out[6]:
```
['sample_submission.csv', 'test.csv', 'train.csv']
```

In [7]:
```
with zipfile.ZipFile(zip_file, 'r') as zip_ref:
    zip_ref.extractall(INPUT_PATH)
#     os.remove()
```

In [8]:
```
os.listdir(INPUT_PATH)
```

Out[8]:
```
['sample_submission.csv',
 'sample_submission.zip',
 'test.csv',
 'test.zip',
 'train.csv',
 'train.zip']
```

In [9]:
```
for filename in os.listdir(INPUT_PATH):
    if filename.endswith('.zip'):
        with zipfile.ZipFile(f'{INPUT_PATH}/{filename}', 'r') as
zip_ref:
            zip_ref.extractall(INPUT_PATH)
        os.remove(f'{INPUT_PATH}/{filename}')
```

In [10]:
```
os.listdir(INPUT_PATH)
```

Out[10]:
```
['sample_submission.csv', 'test.csv', 'train.csv']
```

Read data

Dataset contains text from works of fiction written by spooky authors of the public domain: Edgar Allan Poe, HP Lovecraft and Mary Shelley. The data was prepared by chunking larger texts into sentences using CoreNLP's MaxEnt sentence tokenizer, so you may notice the odd non-sentence here and there. The objective is to accurately identify the author of the sentences.

In [11]:
```
df = pd.read_csv(f'{INPUT_PATH}/train.csv', index_col='id')
```

In [12]:
```
df.shape
```

Out[12]:
```
(19579, 2)
```

Data fields:

`id` - a unique identifier for each sentence

`text` - some text written by one of the authors

`author` - the author of the sentence (EAP: Edgar Allan Poe, HPL: HP Lovecraft; MWS: Mary Wollstonecraft Shelley)

In [13]:
```
df.head()
```

Out[13]:

	text	author
id		
id26305	This process, however, afforded me no means of...	EAP
id17569	It never once occurred to me that the fumbling...	HPL
id11008	In his left hand was a gold snuff box, from wh...	EAP
id27763	How lovely is spring As we looked from Windsor...	MWS
id12958	Finding nothing else, not even gold, the Super...	HPL

Feature engineering and text processing

Do some feature engineering. This consists of two main parts.

```
Meta features - features that are extracted from the text like number o
f words, number of stop words, number of punctuations etc
Text based features - features directly based on the text / words like
frequency, svd, word2vec etc
```

Meta Features:

Lets start with creating meta featues . The feature list is as follows:

- Number of characters
- Number of words
- Number and fraction of punctuation marks
- Number and fraction of nouns
- Number and fraction of adjectives
- Number and fraction of verbs
- Number and fraction of stopwords
- Number and fraction of unique words

In [14]:
```
# lowercase
df['processed'] = df['text'].apply(lambda x: x.lower())
```

In [15]:
```
# count chars and words
df['n_chars'] = df['processed'].apply(lambda x: len(x))
df['n_words'] = df['processed'].apply(lambda x: len(x.split('
')))
```

In [16]:
```
# count punctuation marks
df['n_punctuation'] = df['processed'].apply(lambda x: len([dig fo
r dig in list(x) if dig in string.punctuation]))
```

In [17]:
```
# remove punctuation marks
df['processed'] = df['processed'].apply(lambda x: ''.join(ch for
ch in x if ch not in string.punctuation))
```

In [18]:
```
df.head()
```

Out[18]:

id	text	author	processed	n_chars	n_words	n_punctuation
id26305	This process, however, afforded me no means of...	EAP	this process however afforded me no means of a...	231	41	7
id17569	It never once occurred to me that the fumbling...	HPL	it never once occurred to me that the fumbling...	71	14	1
id11008	In his left hand was a gold snuff box, from wh...	EAP	in his left hand was a gold snuff box from whi...	200	36	5
id27763	How lovely is spring As we looked from Windsor...	MWS	how lovely is spring as we looked from windsor...	206	34	4
id12958	Finding nothing else, not even gold, the Super...	HPL	finding nothing else not even gold the superin...	174	27	4

In [19]:
```
# count nouns, adjetives and verbs
nouns = ('NN','NNP','NNPS','NNS')
adjectives = ('JJ','JJR','JJS')
verbs = ('VB','VBD','VBG','VBN','VBP','VBZ')

df['n_noun'] = df['processed'].apply(lambda x: sum(np.in1d(np.arr
ay(pos_tag(word_tokenize(x)))[:,1], nouns)))
df['n_adj'] = df['processed'].apply(lambda x: sum(np.in1d(np.arra
y(pos_tag(word_tokenize(x)))[:,1], adjectives)))
df['n_verb'] = df['processed'].apply(lambda x: sum(np.in1d(np.arr
ay(pos_tag(word_tokenize(x)))[:,1], verbs)))
```

In [20]:
```
# count stopwords
eng_stopwords = set(stopwords.words("english"))
df['n_stopwords'] = df['processed'].apply(lambda x: sum(np.in1d(w
ord_tokenize(x), eng_stopwords)))
```

In [21]:
```
# unique words
df['n_unique'] = df['processed'].apply(lambda x: len(set(word_tok
enize(x))))
```

In [22]:
```
# fractions

for count in ['n_noun', 'n_adj', 'n_verb', 'n_stopwords', 'n_uniq
ue']:
    df['fract'+count[1:]] = df[count] / df['n_words']

df['fract_punctuation'] = df['n_punctuation']/df['n_chars']
```

In [23]:
```
df.head()
```

Out[23]:

id	text	author	processed	n_chars	n_words	n_punctuation	n_noun	n_adj
id26305	This process, however, afforded me no means of...	EAP	this process however afforded me no means of a...	231	41	7	12	2
id17569	It never once occurred to me that the fumbling...	HPL	it never once occurred to me that the fumbling...	71	14	1	2	1
id11008	In his left hand was a gold snuff box, from wh...	EAP	in his left hand was a gold snuff box from whi...	200	36	5	10	5
id27763	How lovely is spring As we looked from Windsor...	MWS	how lovely is spring as we looked from windsor...	206	34	4	10	6
id12958	Finding nothing else, not even gold, the Super...	HPL	finding nothing else not even gold the superin...	174	27	4	6	1

EDA

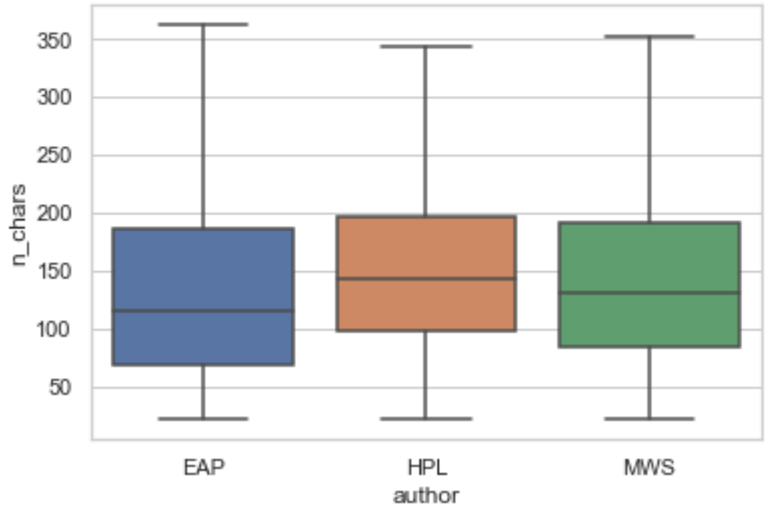
Boxplots

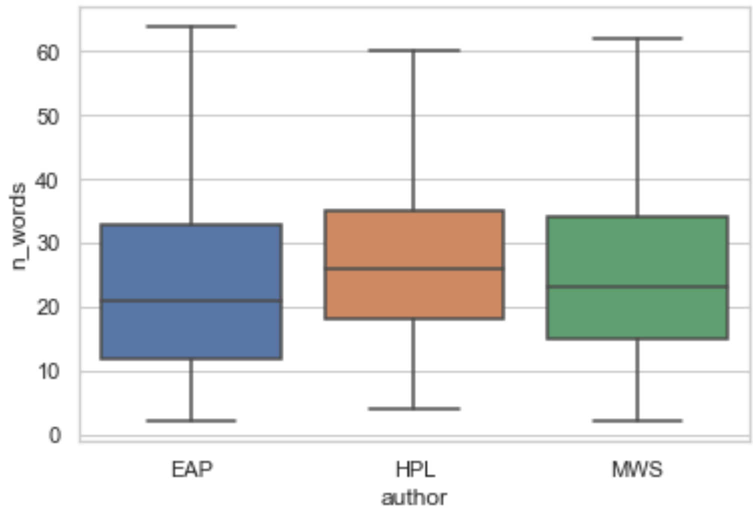
Plot some of new variables to see if they are helpful to predict an author.

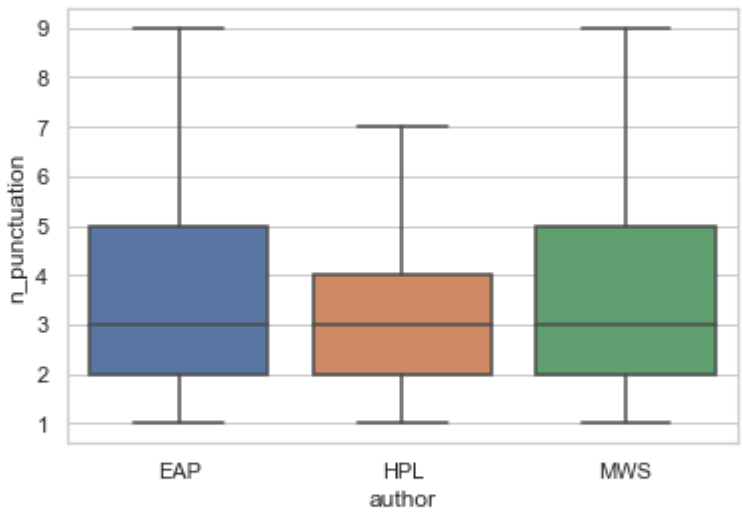
In [24]:
```
cols_plot = list(df.columns[df.columns.get_loc('n_chars'):])
len(cols_plot)
```

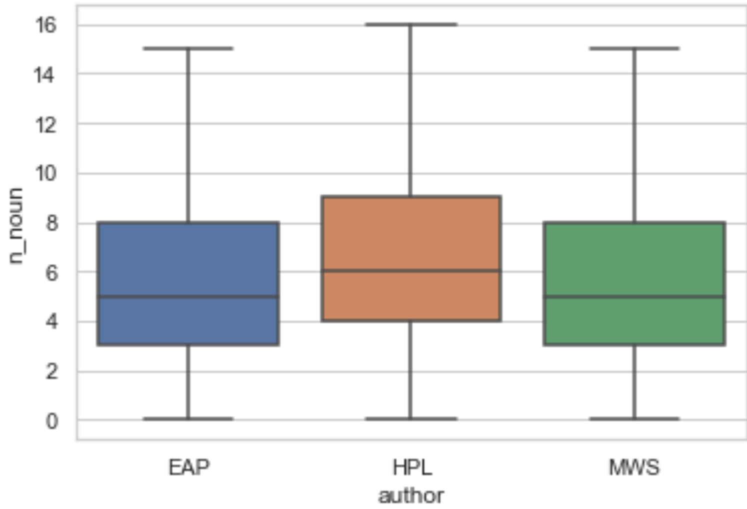
Out[24]: 14

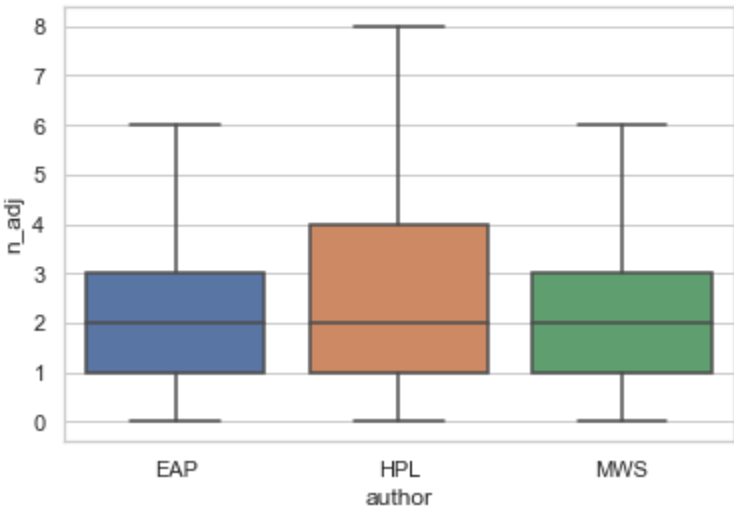
In [25]:
```
for col in list(cols_plot):
    sns.boxplot(x='author', y=col, data=df, showfliers=False)
    plt.show()
    plt.close()
```

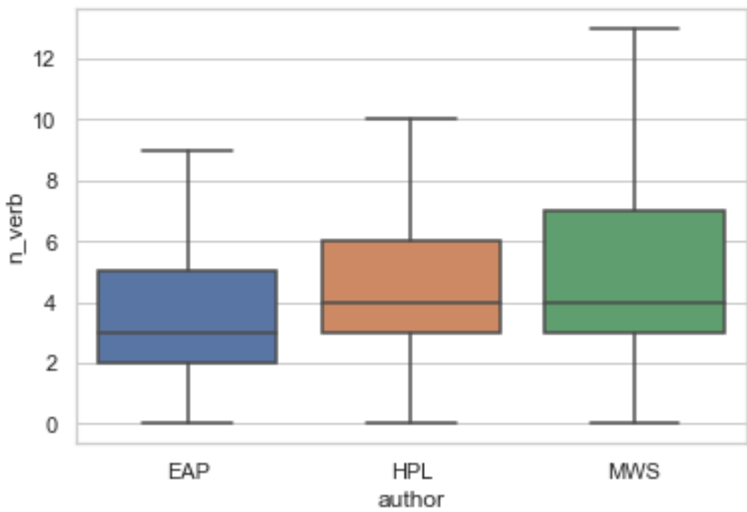

n_chars
350
300
250
200
150
100
50
EAP
HPL
MWS
author

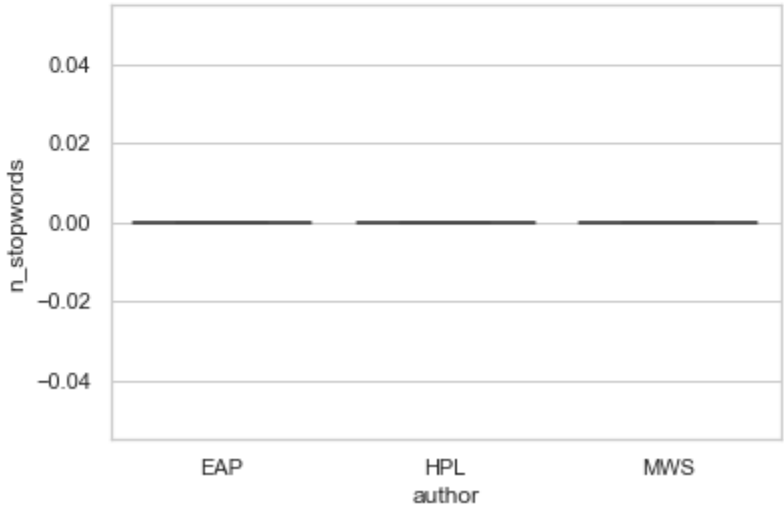

n_words
60
50
40
30
20
10
0
EAP
HPL
MWS
author

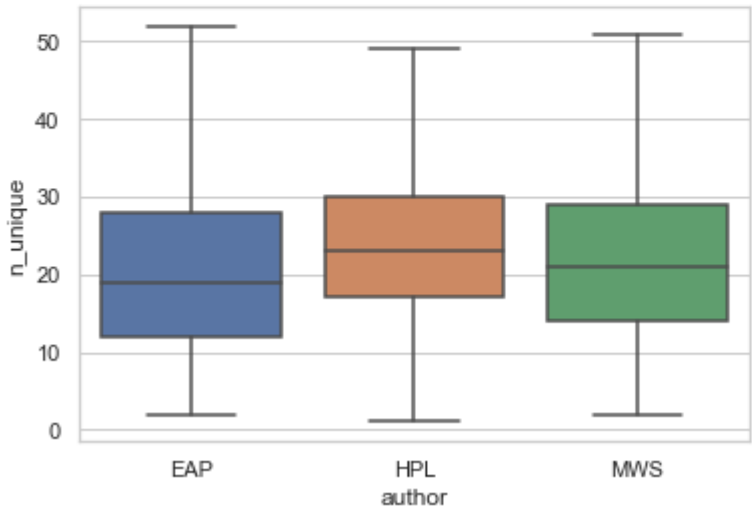

n_punctuation
9
8
7
6
5
4
3
2
1
EAP
HPL
MWS
author

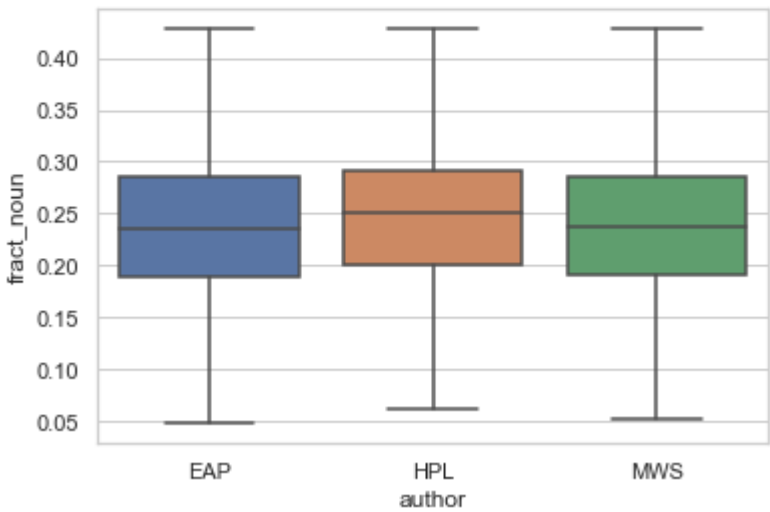

n_noun
16
14
12
10
8
6
4
2
0
EAP
HPL
MWS
author

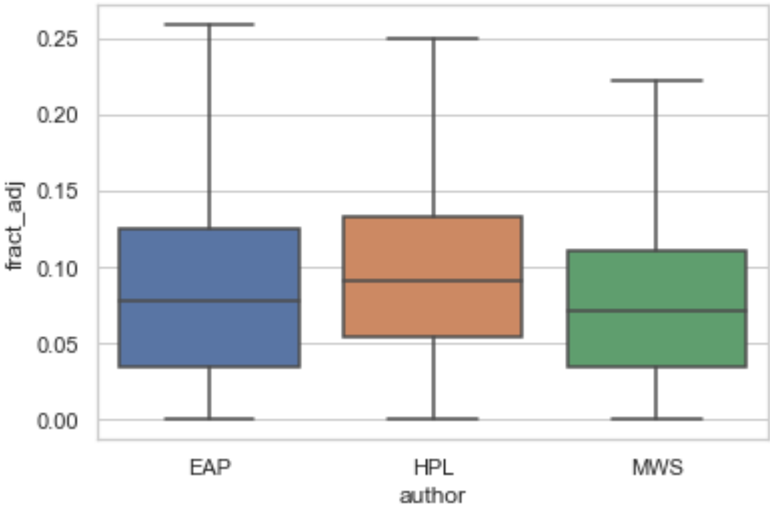

n_adj
8
7
6
5
4
3
2
1
0
EAP
HPL
MWS
author

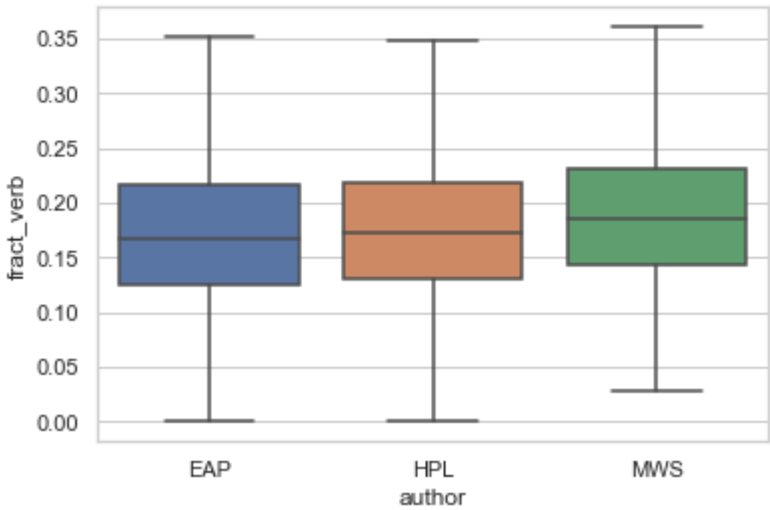

n_verb
12
10
8
6
4
2
0
EAP
HPL
MWS
author

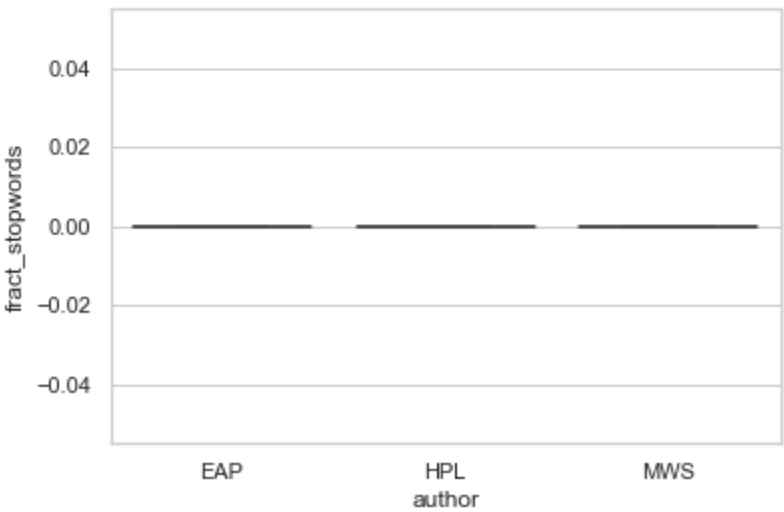

0.04
0.02
0.00
−0.02
−0.04
n_stopwords
EAP
HPL
MWS
author

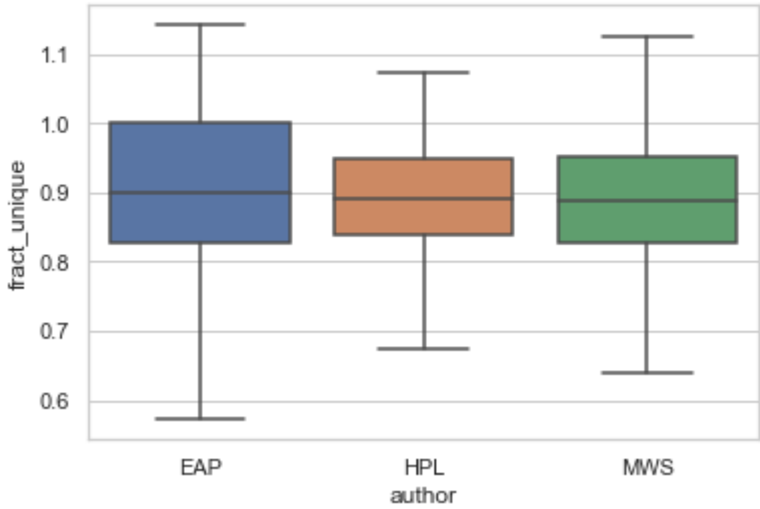

50
40
30
20
10
0
n_unique
EAP
HPL
MWS
author

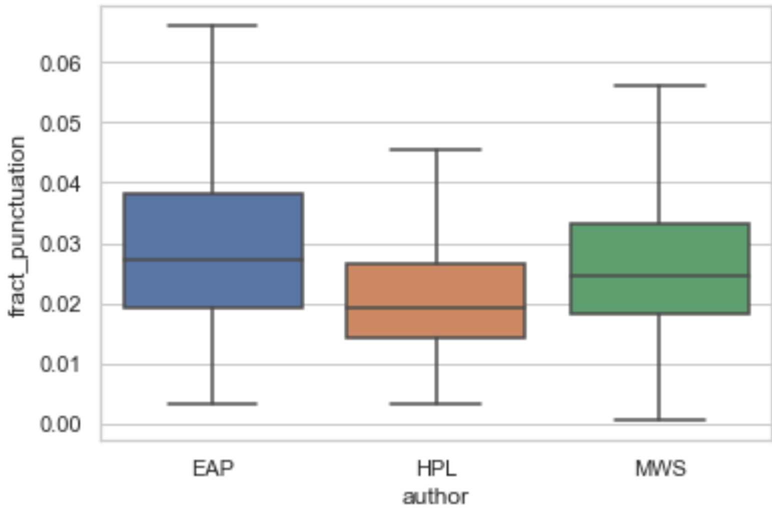

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
fract_noun
EAP
HPL
MWS
author


0.25
0.20
0.15
0.10
0.05
0.00
fract_adj
EAP
HPL
MWS
author


0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
fract_verb
EAP
HPL
MWS
author


0.04
0.02
0.00
−0.02
−0.04
fract_stopwords
EAP
HPL
MWS
author





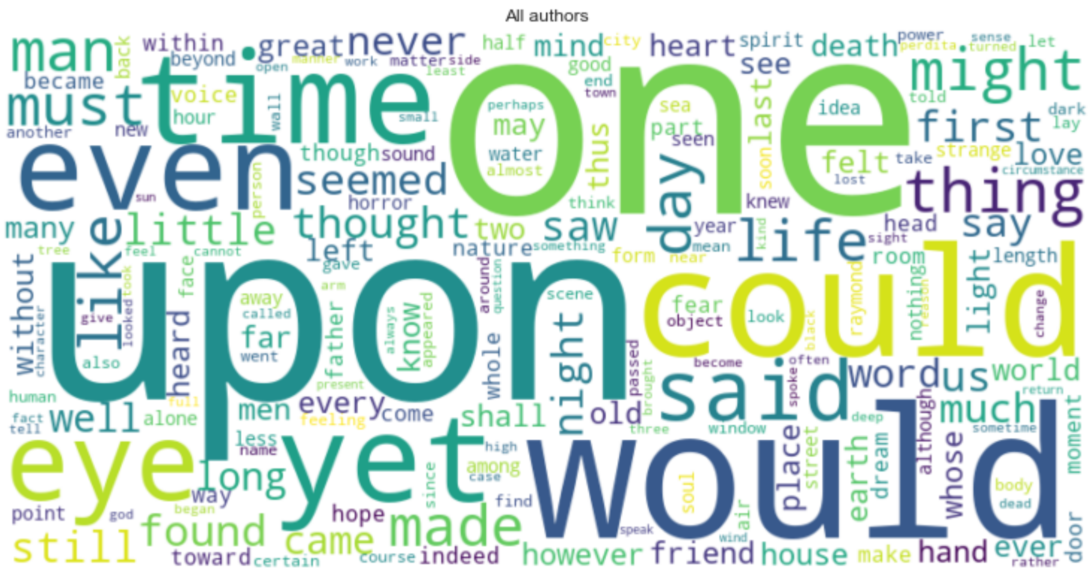
It can be seen that number of nouns, adjectives and verbs, as well as punctuation fraction are slightly different for all authors.

Wordcloud

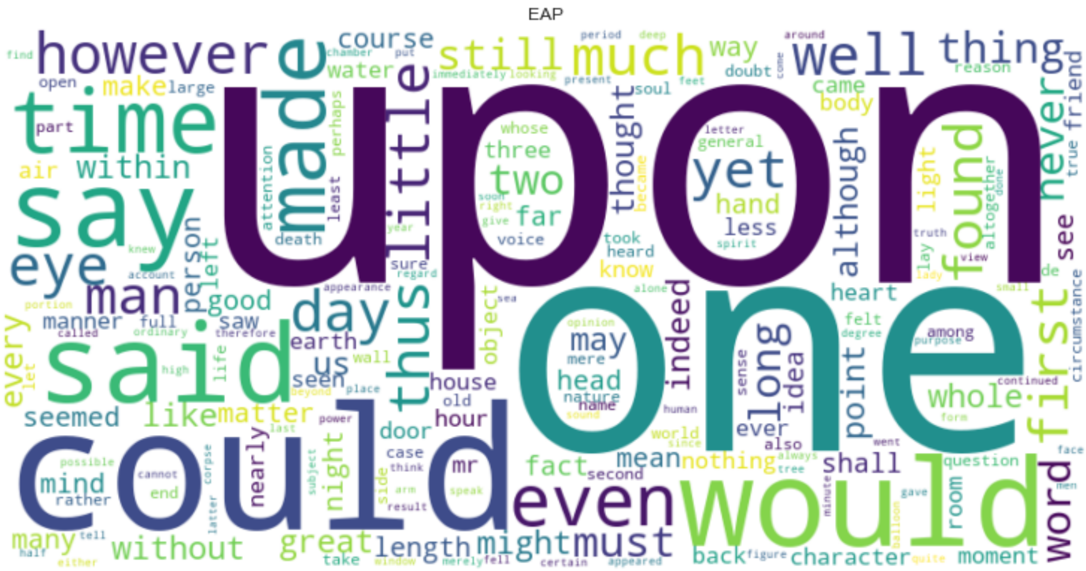
Wordcloud helps us to see the most frequent words in the text in fascinating way. It seems to me that plotting wordcloud of all authors words together and then plotting it for each author would be a good idea. It may give us some hints of how to distinguish between authors by words frequancy only.

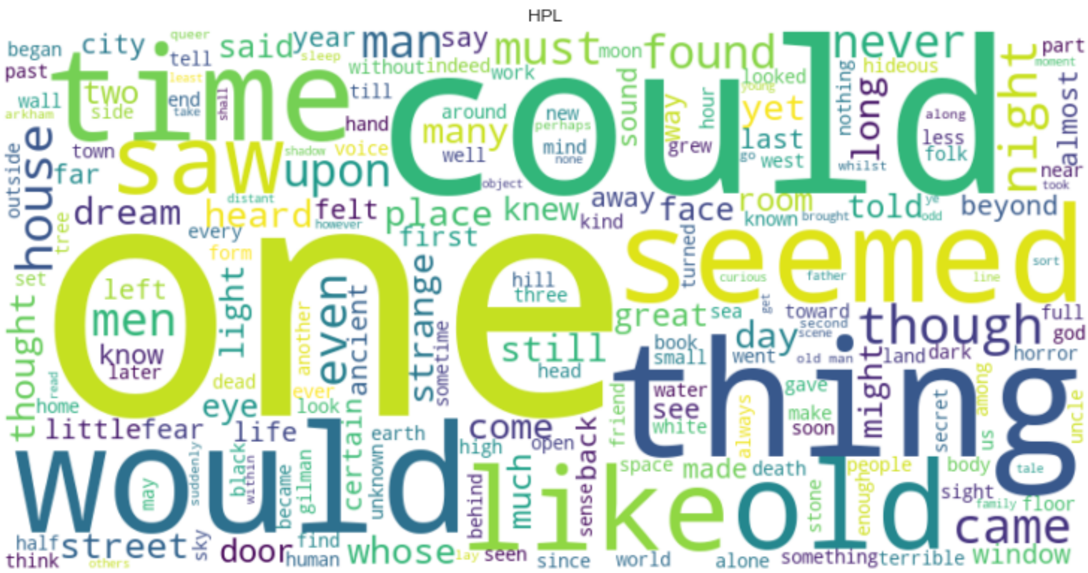
In [26]:
```python
def show_word_cloud(s, author='', save=True):
    text = " ".join(review for review in s)
    # Create stopword list:
    stopwords = set(eng_stopwords)

    # Generate a word cloud image
    wordcloud = WordCloud(width=800, height=400, stopwords=stopwo
rds, background_color="white").generate(text)

    # Display the generated image:
    # the matplotlib way:
    plt.figure(figsize=(13, 10))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.title(author)
    plt.show()
    if save:
#         plt.savefig(f"img/word_cloud{author}.jpg", format="jp
g",dpi=100)
        wordcloud.to_file(f"{IMAGE_PATH}/word_cloud{author}.png")
    plt.close()
```

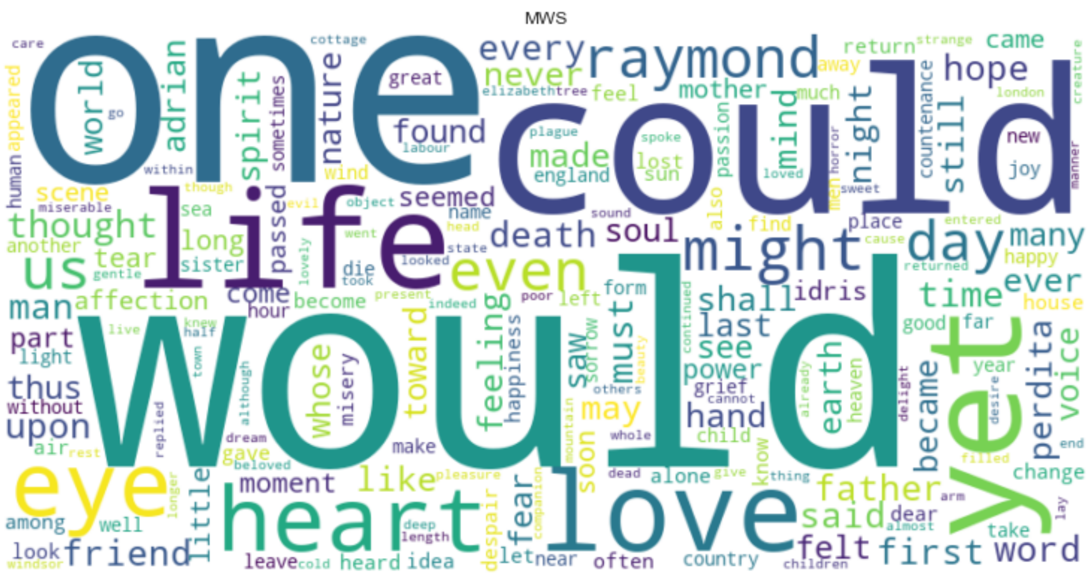
In [27]:
```python
show_word_cloud(df['processed'], 'All authors')
```



In [28]:
```
for author in df['author'].unique():
    show_word_cloud(df[df['author'] == author]['processed'], auth
or)
```







Stemming

In grammar, inflection is the modification of a word to express different grammatical categories such as tense, case, voice, aspect, person, number, gender, and mood. An inflection expresses one or more grammatical categories with a prefix, suffix or infix, or another internal modification such as a vowel change.

Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language.

Stem (root) is the part of the word to which you add inflectional (changing/deriving) affixes such as (-ed,-ize, -s,-de,mis). So stemming a word or sentence may result in words that are not actual words. Stems are created by removing the suffixes or prefixes used with a word.

In [29]:
```
stemmer=SnowballStemmer("english")

def stem(s):
    return ' '.join([stemmer.stem(word) for word in word_tokenize
(s)])

%time df['processed'] = df['processed'].apply(lambda x: stem(x))
```

```
CPU times: user 5.61 s, sys: 0 ns, total: 5.61 s
Wall time: 5.61 s
```

In [30]:
```
df.head()
```

Out[30]:

id	text	author	processed	n_chars	n_words	n_punctuation	n_noun	n_adj
id26305	This process, however, afforded me no means of...	EAP	this process howev afford me no mean of ascert...	231	41	7	12	2
id17569	It never once occurred to me that the fumbling...	HPL	it never onc occur to me that the fumbl might ...	71	14	1	2	1
id11008	In his left hand was a gold snuff box, from wh...	EAP	in his left hand was a gold snuff box from whi...	200	36	5	10	5
id27763	How lovely is spring As we looked from Windsor...	MWS	how love is spring as we look from windsor ter...	206	34	4	10	6
id12958	Finding nothing else, not even gold, the Super...	HPL	find noth els not even gold the superintend ab...	174	27	4	6	1

Train/test split

Simple 80/20 split

In [31]:
```
train_cols = list(df.columns[df.columns.get_loc('processed'):])
X, y = df[train_cols].copy(), df['author'].copy()
```

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_si
ze=0.2, random_state=42)

X_train = X_train.copy()
X_test = X_test.copy()
```

TF-IDF

In a large text corpus, some words will be very present (e.g. “the”, “a”, “is” in English) hence carrying very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms.

In order to re-weight the count features into floating point values suitable for usage by a classifier it is very common to use the tf–idf transform.

Tf means **term-frequency** while tf–idf means term-frequency times **inverse document-frequency**.

In [33]:
```
tfidf = TfidfVectorizer(stop_words=eng_stopwords, min_df=3)
tfidf.fit(X_train['processed'])
X_train = np.concatenate([X_train, tfidf.transform(X_train['processed']).toarray()], axis=1)
X_test = np.concatenate([X_test, tfidf.transform(X_test['processed']).toarray()], axis=1)
```

In [34]:
```
X_train.shape
```

Out[34]:
```
(15663, 7406)
```

In [35]:
```
# drop the first columns which is 'processed'
X_train, X_test = X_train[:, 1:], X_test[:, 1:]
```

Model

Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the “naive” assumption of conditional independence between every pair of features given the value of the class variable.

MultinomialNB implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice).

GridSearchCV is used to find optimal α for Naive Bayes Classificator.

In [36]:
```
gs = GridSearchCV(
    MultinomialNB(),
    param_grid = { 'alpha':(0.001, 0.01,0.05, 0.1, 0.5, 1, 10)},
    scoring='neg_log_loss',
    n_jobs = 1,
    cv=4,
    verbose=100,
    refit=True
)
gs.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 7 candidates, totalling 28 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 conc
urrent workers.
[CV] alpha=0.001
....................................................
[CV] ....................... alpha=0.001, score=-0.472, total=
1.9s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.3s rema
ining:    0.0s
[CV] alpha=0.001
....................................................
[CV] ....................... alpha=0.001, score=-0.496, total=
1.9s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    4.6s rema
ining:    0.0s
[CV] alpha=0.001
....................................................
[CV] ....................... alpha=0.001, score=-0.482, total=
1.9s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    6.9s rema
ining:    0.0s
[CV] alpha=0.001
....................................................
[CV] ....................... alpha=0.001, score=-0.495, total=
1.9s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    9.2s rema
ining:    0.0s
[CV] alpha=0.01
.....................................................
[CV] ........................ alpha=0.01, score=-0.435, total=
1.9s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   11.5s rema
ining:    0.0s
[CV] alpha=0.01
.....................................................
[CV] ........................ alpha=0.01, score=-0.453, total=
2.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:   13.8s rema
ining:    0.0s
[CV] alpha=0.01
.....................................................
[CV] ........................ alpha=0.01, score=-0.444, total=
2.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:   16.1s rema
ining:    0.0s
[CV] alpha=0.01
.....................................................
[CV] ........................ alpha=0.01, score=-0.446, total=
2.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:   18.5s rema
ining:    0.0s
[CV] alpha=0.05
.....................................................
[CV] ........................ alpha=0.05, score=-0.437, total=
2.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:   20.8s rema
ining:    0.0s
[CV] alpha=0.05
.....................................................
```

```
[CV] ........................ alpha=0.05, score=-0.450, total=
2.0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:   23.1s rema
ining:    0.0s
[CV] alpha=0.05
.......................................................
[CV] ........................ alpha=0.05, score=-0.447, total=
2.0s
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:   25.5s rema
ining:    0.0s
[CV] alpha=0.05
.......................................................
[CV] ........................ alpha=0.05, score=-0.440, total=
2.0s
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:   27.8s rema
ining:    0.0s
[CV] alpha=0.1
........................................................
[CV] ......................... alpha=0.1, score=-0.450, total=
2.0s
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:   30.1s rema
ining:    0.0s
[CV] alpha=0.1
........................................................
[CV] ......................... alpha=0.1, score=-0.461, total=
2.0s
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:   32.5s rema
ining:    0.0s
[CV] alpha=0.1
........................................................
[CV] ......................... alpha=0.1, score=-0.460, total=
2.0s
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:   34.8s rema
ining:    0.0s
[CV] alpha=0.1
........................................................
[CV] ......................... alpha=0.1, score=-0.450, total=
2.0s
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:   37.1s rema
ining:    0.0s
[CV] alpha=0.5
........................................................
[CV] ......................... alpha=0.5, score=-0.524, total=
2.0s
[Parallel(n_jobs=1)]: Done  17 out of  17 | elapsed:   39.5s rema
ining:    0.0s
[CV] alpha=0.5
........................................................
[CV] ......................... alpha=0.5, score=-0.529, total=
2.0s
[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed:   41.8s rema
ining:    0.0s
[CV] alpha=0.5
........................................................
[CV] ......................... alpha=0.5, score=-0.536, total=
2.0s
[Parallel(n_jobs=1)]: Done  19 out of  19 | elapsed:   44.1s rema
ining:    0.0s
[CV] alpha=0.5
........................................................
```

```
[CV] ........................... alpha=0.5, score=-0.520, total=
1.9s
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:   46.4s rema
ining:    0.0s
[CV] alpha=1
..........................................................
[CV] ............................ alpha=1, score=-0.583, total=
1.9s
[Parallel(n_jobs=1)]: Done  21 out of  21 | elapsed:   48.7s rema
ining:    0.0s
[CV] alpha=1
..........................................................
[CV] ............................ alpha=1, score=-0.586, total=
1.9s
[Parallel(n_jobs=1)]: Done  22 out of  22 | elapsed:   51.0s rema
ining:    0.0s
[CV] alpha=1
..........................................................
[CV] ............................ alpha=1, score=-0.597, total=
1.9s
[Parallel(n_jobs=1)]: Done  23 out of  23 | elapsed:   53.3s rema
ining:    0.0s
[CV] alpha=1
..........................................................
[CV] ............................ alpha=1, score=-0.579, total=
1.9s
[Parallel(n_jobs=1)]: Done  24 out of  24 | elapsed:   55.6s rema
ining:    0.0s
[CV] alpha=10
.........................................................
[CV] ........................... alpha=10, score=-1.990, total=
2.0s
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:   57.9s rema
ining:    0.0s
[CV] alpha=10
.........................................................
[CV] ........................... alpha=10, score=-1.922, total=
1.9s
[Parallel(n_jobs=1)]: Done  26 out of  26 | elapsed:  1.0min rema
ining:    0.0s
[CV] alpha=10
.........................................................
[CV] ........................... alpha=10, score=-2.003, total=
1.9s
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:  1.0min rema
ining:    0.0s
[CV] alpha=10
.........................................................
[CV] ........................... alpha=10, score=-1.991, total=
1.9s
[Parallel(n_jobs=1)]: Done  28 out of  28 | elapsed:  1.1min rema
ining:    0.0s
[Parallel(n_jobs=1)]: Done  28 out of  28 | elapsed:  1.1min fini
```

Out[36]:
```
GridSearchCV(cv=4, estimator=MultinomialNB(), n_jobs=1,
             param_grid={'alpha': (0.001, 0.01, 0.05, 0.1, 0.5,
1, 10)},
             scoring='neg_log_loss', verbose=100)
```

In [37]:
```
nb = gs.best_estimator_
nb
```

Out[37]:
```
MultinomialNB(alpha=0.05)
```

In [38]:
```
accuracy_score(y_test, nb.predict(X_test))
```

Out[38]:
```
0.8215015321756894
```

Random Forest

In [39]:
```
rf_params = dict(max_depth=710, max_features=0.0225716742746937,
                       max_samples=0.7091283290110244, min_sample
s_leaf=4,
                       min_samples_split=6)
rf_params['n_estimators'] = 200
rf_params['random_state'] = 42
rf_params['n_jobs'] = -1
rf = RandomForestClassifier(**rf_params)
rf.fit(X_train, y_train)
```

Out[39]:
```
RandomForestClassifier(max_depth=710, max_features=0.022571674274
6937,
                       max_samples=0.7091283290110244, min_sample
s_leaf=4,
                       min_samples_split=6, n_estimators=200, n_j
obs=-1,
                       random_state=42)
```

In [40]:
```
accuracy_score(y_test, rf.predict(X_test))
```

Out[40]:
```
0.6422369765066395
```

Unfortunately, even after tunning the score is low :(

SVM

In [41]:
```
svm_pipe= Pipeline([
    ('scaler', MinMaxScaler()),
    ('svm', SVC(max_iter=1200, random_state=42))
])

%time svm_pipe.fit(X_train, y_train)
```

```
CPU times: user 9min 36s, sys: 309 ms, total: 9min 36s
Wall time: 9min 37s

/home/daryna/anaconda3/envs/ml_ukma/lib/python3.7/site-packages/s
klearn/svm/_base.py:249: ConvergenceWarning: Solver terminated ea
rly (max_iter=1200).  Consider pre-processing your data with Stan
dardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

Out[41]:
```
Pipeline(steps=[('scaler', MinMaxScaler()),
                ('svm', SVC(max_iter=1200, random_state=42))])
```

In [42]:
```
accuracy_score(y_test, svm_pipe.predict(X_test))
```

Out[42]:
```
0.7637895812053116
```

Voting Classifier

Now lets ensemble Naive Bayes, Random Forest and SVM classifiers using Voting Classifier.

P.S. Unfortunately, stacking leads to memory error, thus I can't use it :(

In [43]:
```
vote_clf = VotingClassifier(
    estimators = [
        ('nb', nb),
        ('rf', rf),
        ('svm', svm_pipe)
    ],
#     final_estimator=LogisticRegression(),
    n_jobs=-1,
#     cv=3,
#     verbose=10
)
vote_clf.fit(X_train, y_train)
```

Out[43]:
```
VotingClassifier(estimators=[('nb', MultinomialNB(alpha=0.05)),
                             ('rf',
                              RandomForestClassifier(max_depth=71
0,
                                                     max_features
=0.0225716742746937,
                                                     max_samples=
0.7091283290110244,
                                                     min_samples_
leaf=4,
                                                     min_samples_
split=6,
                                                     n_estimators
=200,
                                                     n_jobs=-1,
                                                     random_state
=42)),
                             ('svm',
                              Pipeline(steps=[('scaler', MinMaxSc
aler()),
                                              ('svm',
                                               SVC(max_iter=1200,
                                                   random_state=4
2))]))],
                 n_jobs=-1)
```

In [44]:
```
accuracy_score(y_test, vote_clf.predict(X_test))
```

Out[44]:
```
0.8102655771195098
```

In [45]:
```
vote_clf.estimators_
```

Out[45]:
```
[MultinomialNB(alpha=0.05),
 RandomForestClassifier(max_depth=710, max_features=0.02257167427
46937,
                        max_samples=0.7091283290110244, min_sampl
es_leaf=4,
                        min_samples_split=6, n_estimators=200, n_
jobs=-1,
                        random_state=42),
 Pipeline(steps=[('scaler', MinMaxScaler()),
                 ('svm', SVC(max_iter=1200, random_state=42))])]
```

In [46]:
```
for clf in [nb, rf, svm_pipe, vote_clf]:
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
MultinomialNB 0.8215015321756894
RandomForestClassifier 0.6422369765066395
Pipeline 0.7637895812053116
VotingClassifier 0.8102655771195098
```

In [47]:
```
df.shape
```

Out[47]:
```
(19579, 17)
```

VotingClassifier does not show best results. Random Forest has quite low accuracy comparatively to Naive Bayes and SVM, which is likely to be the reason of Voting Classifier to behave this way.

Summary

1. Text Processing.

```
1) Lowercase
2) Remove punctuation
3) Remove stopwords
4) Stemming
```

2. Feature Egineering.

```
- Meta features - number/fraction of words/characters
- Text based features - tf-idf
```

3. Train/test split.

```
Train on 80% of data and test on 20% of data.
```

4. Model

```
- Naive Bayes------|
- Random Forest----|===> VotingClassifier
- SVM--------------|
```

What else to try?

- word2vec (Bogdan mentioned it in his presentation)
- stacking (unfortunately not on this computer :(
- cross validation
- removing Random Forest from ensemble

In [ ]: