Group Project Report

Spooky Author Identification

CPSC 483

Prepared by:

Anoop Jeerige

Rounak Kulkarni

Sanika Deshpande

Sweety Jain

Mohit Kirange

Contents

[1. Introduction 4](#_Toc501026648)

[2. Competition Overview 4](#_Toc501026649)

[3. Data 4](#_Toc501026650)

[4. Environment 4](#_Toc501026651)

[5. Exploratory Data Analysis 1 4](#_Toc501026652)

[5.1. Basic Statistics across authors 5](#_Toc501026653)

[5.1.1. Sentence distribution 5](#_Toc501026654)

[5.1.2. Common words 6](#_Toc501026655)

[5.1.3. Characteristic words 8](#_Toc501026656)

[5.2. Conclusion 8](#_Toc501026657)

[6. Bag of Words Approach 9](#_Toc501026658)

[6.1. Count Vectorizer 9](#_Toc501026659)

[6.2. Tf-idf term weighting 9](#_Toc501026660)

[6.3. Spooky Author Identification Analysis 10](#_Toc501026661)

[6.3.1. Load the data 10](#_Toc501026662)

[6.3.2. Training and Testing data split 10](#_Toc501026663)

[6.3.3. Extracting features from text – Bag of Words 11](#_Toc501026664)

[6.3.3.1. Tokenizing 11](#_Toc501026665)

[6.3.3.2. Occurrences to frequencies 11](#_Toc501026666)

[6.3.3.3. Train a classifier 11](#_Toc501026667)

[6.3.3.4. Evaluation of performance on test set 12](#_Toc501026668)

[6.3.4. Pipeline 13](#_Toc501026669)

[6.3.5. Benchmarking classifiers 13](#_Toc501026670)

[6.3.6. Parameter tuning using grid search 14](#_Toc501026671)

[6.3.7. Submission and Scores 15](#_Toc501026672)

[6.4. Limitations 15](#_Toc501026673)

[7. Exploratory Data Analysis 2 15](#_Toc501026674)

[7.1. Grammar: Parts of Speech 16](#_Toc501026675)

[7.2. Conclusion 16](#_Toc501026676)

[8. Bag of Words Improvements 16](#_Toc501026677)

[8.1. Custom token pattern and binary occurrence marker 16](#_Toc501026678)

[8.2. Customer tokenizer 17](#_Toc501026679)

[9. Adhoc Experiments 18](#_Toc501026680)

[9.1. AutoML 18](#_Toc501026681)

[9.2. VotingClassifier 18](#_Toc501026682)

[10. References 19](#_Toc501026683)

# **Introduction**

The purpose of this document is to capture the details of the data mining process followed for the Kaggle competition – Spooky Author Identification.

# **Competition Overview**

* **Challenge:** To accurately identify the author of the sentences in the test set.
* **Evaluation:** Based on multi-class logarithmic loss.
* **Submission:** csv file with the id and a probability for each of the three classes

# **Data**

The dataset contains text from works of fiction written by spooky authors of the public domain: Edgar Allan Poe, HP Lovecraft, and Mary Shelley.

The dataset includes three files –

1. train.csv - the training set
2. test.csv - the test set
3. sample\_submission.csv - a sample submission file in the correct format

**File Descriptions**

Data Fields (train.csv)

* id - a unique identifier for each sentence
* text - some text written by one of the authors
* Author(label) - the author of the sentence (EAP: Edgar Allan Poe, HPL: HP Lovecraft; MWS: Mary Wollstonecraft Shelley)

Data Fields (test.csv)

* id - a unique identifier for each sentence
* text - some text written by one of the authors

# **Environment**

The development environment will be setup on the Windows10 platform, and coding will be done in Python 3.6 on Jupyter Notebook. The scikit-learn framework will be used to perform data mining and data analysis tasks, and other supporting libraries include pandas, numpy, matplotlib, and nltk.

# **Exploratory Data Analysis 1**

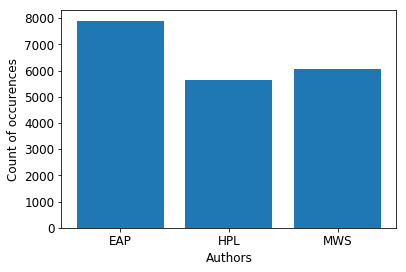
**Notebook Reference:** 1\_Exploratory Data Analysis

Exploratory Data Analysis aims to find patterns and insights is data, which helps to guide the analytics and machine learning experiments to be performed on the data.

# **Basic Statistics across authors**

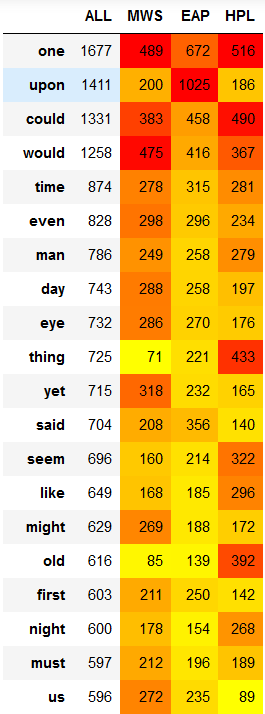
In context of the Spooky Author Identification basic stats will be extracted from the sentences such as sentence distribution, common words and characteristic words.

# Sentence distribution

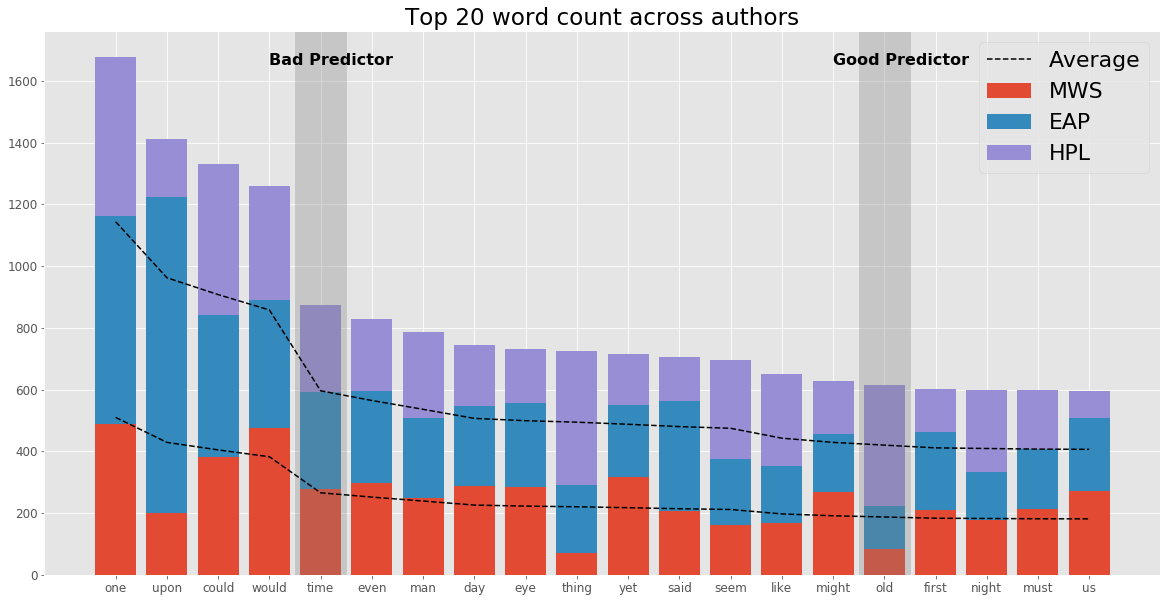


The above graph shows the sentences distributed across the three authors. There is not much imbalance in the distribution across the three author classes. Hence train\_test\_split the can be used for the splitting of the data into training and testing sets. In case of imbalanced classes we can use sklearn.model\_selection.StratifiedKFold to preserve percentage of samples of each class

# **Common words**

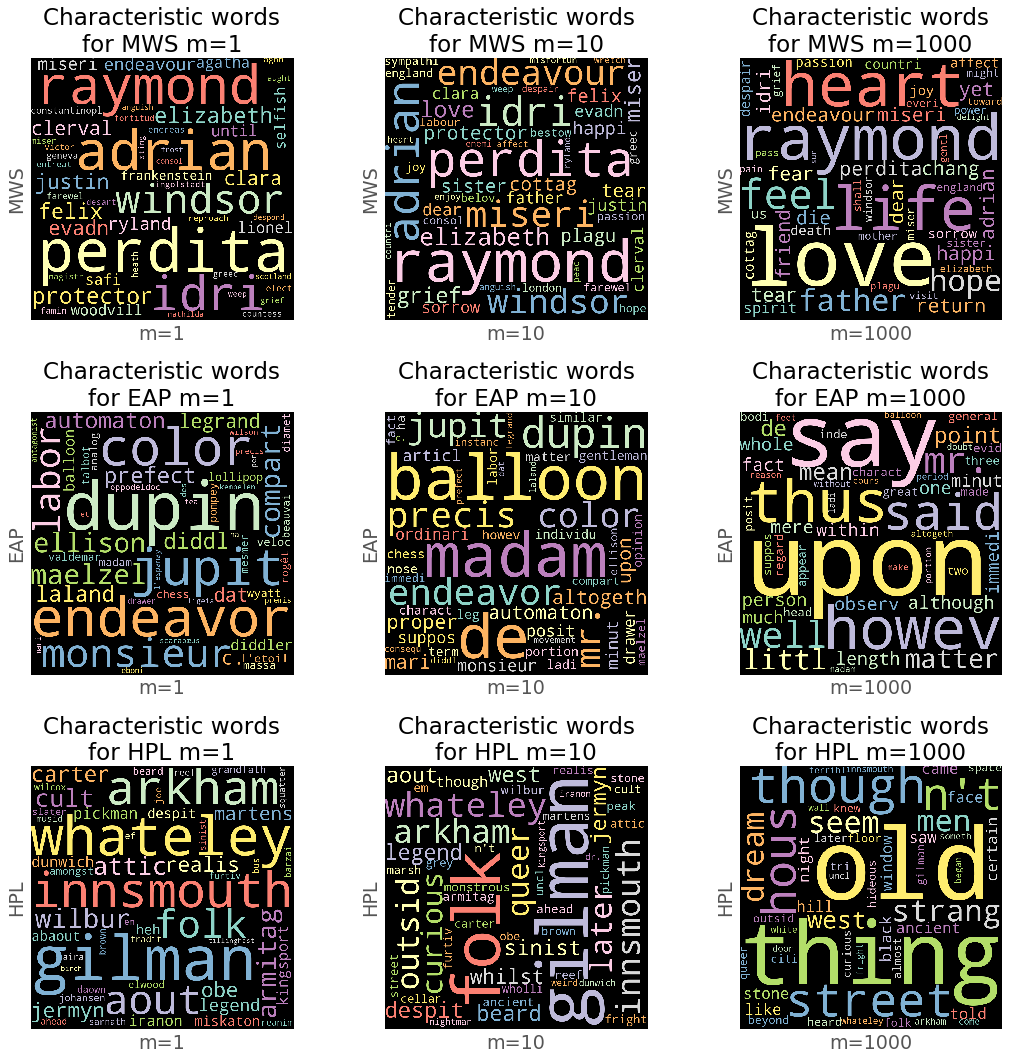


The above table shows the top 20 common words used and their frequencies across the three authors.



The same table can be plotted as a graph to enable better visualization of the common words. Words like ‘time’ that have equal distribution across the authors are bad predictors whereas words like ‘old’ serve as good predictors.

# **Characteristic words**



The above graph shows the distribution of the characteristic words across the three authors where m denotes the frequency of the word occurrence.

# **Conclusion**

The initial data exploration shows that there is considerable predictable power in the words across the sentences of the three authors. Hence the Bag of approach would be a good straight forward approach to begin with.

# **Bag of Words Approach**

**Notebook reference: 2\_Bag of Words**

Machine learning algorithms generally expect numerical feature vectors with fixed size rather than raw text of variable length. Bag of Words is a general process of converting text data into numerical feature vectors. It involves tokenization, counting and normalization.

1. **Tokenization**

Tokenizing strings and assigning an integer id for each possible token. White spaces or punctuation can be used as token separators

1. **Counting**

Counting the occurrences of the token in each sentence

1. **Normalization**

Normalizing and weighting with diminishing importance of tokens that occur in most sentences

So, in Bag of Words approach, features are defined as:

* Each individual token occurrence frequency (normalized or not) is a feature.
* The vector of all the token frequencies for a given sentence is considered a multivariate sample.

# **Count Vectorizer**

CountVectorizer of scikit-learn implements both tokenization and occurrence counting in a single class. The model has various parameters that enables us to tune the vectorization of text sentences into numerical vectors as required.

CountVectorizer(analyzer=...'word', binary=False, decode\_error=...'strict',

dtype=<... 'numpy.int64'>, encoding=...'utf-8', input=...'content',

lowercase=True, max\_df=1.0, max\_features=None, min\_df=1,

ngram\_range=(1, 1), preprocessor=None, stop\_words=None,

strip\_accents=None, token\_pattern=...'(?u)\\b\\w\\w+\\b',

tokenizer=None, vocabulary=None)

# **Tf-idf term weighting**

Term-frequency features are computed by dividing the number of occurrences of each word in the sentence by the total number of words in the sentence. Followed by downscaling weights for words that occur in many sentences, as they are less informative than those that occur in a small portion of the corpus. This downscaling is called tf-idf - Term-frequency times inverse document-frequency.

Term-frequency times inverse document-frequency re-weights the count features into floating point values that are suitable for usage by a classifier.

Tf-idf mainly caters to issue where longer sentences will have higher average count values than shorter sentences.

TfidfTransformer(norm=...'l2', smooth\_idf=False, sublinear\_tf=False,

use\_idf=True)

# **Spooky Author Identification Analysis**

**Notebook reference: - 2\_Bag of Words notebook**

Spooky Author Identification analysis details the steps of initial author data preparation, the application of bag of words approach to the author data, experiments with various classifiers, parameter tuning and submission of results to Kaggle to obtain the scores.

# **Load the data**

The training and test csv files will be loaded into pandas dataframe and used for further processing.

# Set the path of the spooky author dataset

SPOOKY\_PATH = 'spooky'

def load\_spooky\_dataset(dataset\_type):

filepath = os.path.join(SPOOKY\_PATH, dataset\_type, dataset\_type + ".csv")

print(filepath)

return pd.read\_csv(filepath))

# **Training and Testing data split**

The train data will be split using scikit-learn function train\_test\_split to get the training and testing datasets.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(train['text'], train['author'])

(14684,) (4895,) (14684,) (4895,)

# **Extracting features from text – Bag of Words**

# **Tokenizing**

count\_vect = CountVectorizer(stop\_words='english')

X\_train\_counts = count\_vect.fit\_transform(X\_train)

X\_train\_counts.shape, X\_train.shape

((14684, 22082), (14684,))

# **Occurrences to frequencies**

tf\_transformer = TfidfTransformer()

X\_train\_tfidf = tf\_transformer.fit\_transform(X\_train\_counts)

X\_train\_tfidf.shape

(14684, 22082)

print(X\_train\_tfidf[0])

(0, 4778) 0.251038917494

(0, 4775) 0.393671188645

(0, 19565) 0.255879097696

(0, 7101) 0.456917074663

(0, 11295) 0.281321257897

(0, 15111) 0.382529390885

(0, 5694) 0.531304758441

# **Train a classifier**

clf = MultinomialNB()

clf.fit(X\_train\_tfidf, y\_train)

MultinomialNB(alpha=1.0, class\_prior=None, fit\_prior=True)

# **Evaluation of performance on test set**

# Prediction on test set

X\_test\_counts = count\_vect.transform(X\_test)

X\_test\_tfidf = tf\_transformer.transform(X\_test\_counts)

X\_test.shape, X\_test\_tfidf.shape

((4895,), (4895, 22082))

y\_pred = clf.predict(X\_test\_tfidf)

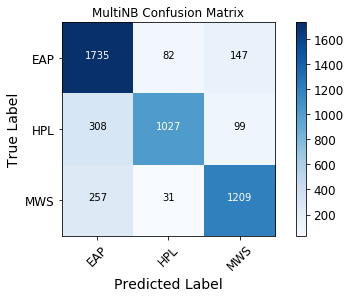
y\_pred\_prob = clf.predict\_proba(X\_test\_tfidf)

print("Accuracy Score:", accuracy\_score(y\_test, y\_pred))

print("Log Loss:", log\_loss(y\_test, y\_pred\_prob))

Accuracy Score: 0.811235955056

Log Loss: 0.604572239607



print(classification\_report(y\_test, y\_pred))

precision recall f1-score support

EAP 0.75 0.88 0.81 1964

HPL 0.90 0.72 0.80 1434

MWS 0.83 0.81 0.82 1497

avg / total 0.82 0.81 0.81 4895

# **Pipeline**

# Pipeline features - vectorizer -> transformer -> classifier

# CountVectorizer - Text Preprocessing, tokenizing, and filtering of stopwords - dictionary of feature indices

# TfidfTransformer - Term frequencies, and downscaling weights for words - tf and tf-idf

# MultinomialNB - Naive Bayes Classifier - multinomial variant classifier

mnb\_clf = Pipeline([

('vect', CountVectorizer()),

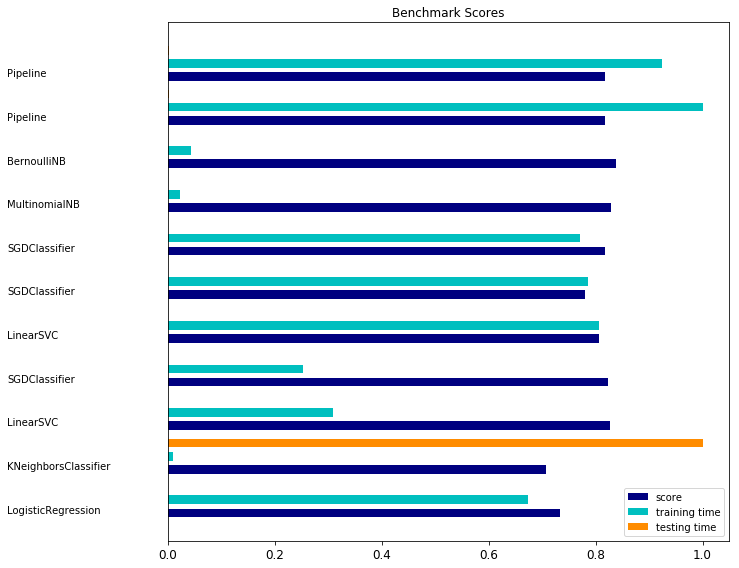
('tfidf', TfidfTransformer()),

('clf', MultinomialNB()),

])

# **Benchmarking classifiers**

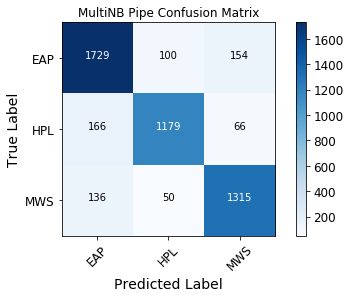
Benchmarking of various classifiers using the bag of words approach applied to the author dataset.



The reports generated as a part of benchmarking are attached below.



# **Parameter tuning using grid search**



parameters = {

'vect\_\_max\_df': (0.5, 0.75, 1.0),

'vect\_\_max\_features': (None, 5000, 10000, 50000),

'vect\_\_ngram\_range': ((1, 1), (1, 2)),

'tfidf\_\_use\_idf': (True, False),

'tfidf\_\_norm': ('l1', 'l2'),

'clf\_\_alpha': (1e-2, 1e-3),

}

mnb\_gs\_clf = GridSearchCV(mnb\_clf, parameters, n\_jobs=-1)

mnb\_gs\_clf.fit(X\_train, y\_train)

mnb\_gs\_clf.best\_score\_

0.83594388450013624

#Best params

for param\_name in sorted(parameters.keys()):

print("%s: %r" % (param\_name, mnb\_gs\_clf.best\_params\_[param\_name]))

clf\_\_alpha: 0.01

tfidf\_\_norm: 'l2'

tfidf\_\_use\_idf: False

vect\_\_max\_df: 0.5

vect\_\_max\_features: None

vect\_\_ngram\_range: (1, 2)

y\_pred = mnb\_gs\_clf.best\_estimator\_.predict(X\_test)

y\_pred\_prob = mnb\_gs\_clf.best\_estimator\_.predict\_proba(X\_test)

print("Accuracy score:", accuracy\_score(y\_test, y\_pred))

print("Log loss:", log\_loss(y\_test, y\_pred\_prob))

Accuracy score: 0.862717058223

Log loss: 0.354775708031

# **Submission and Scores**

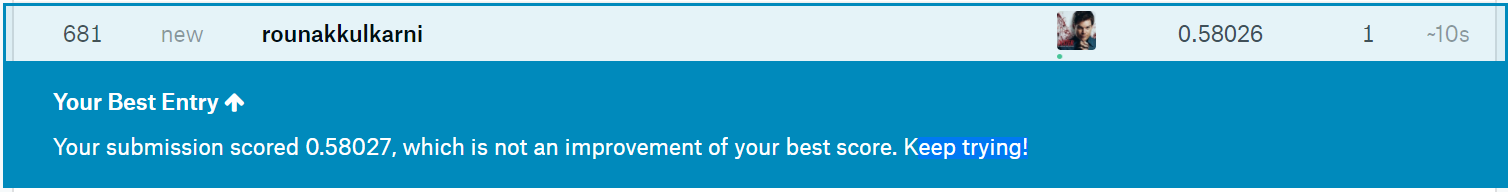
def create\_submission\_file(y\_pred\_prob, name):

result = pd.DataFrame(y\_pred\_prob, columns=['EAP', 'HPL', 'MWS'])

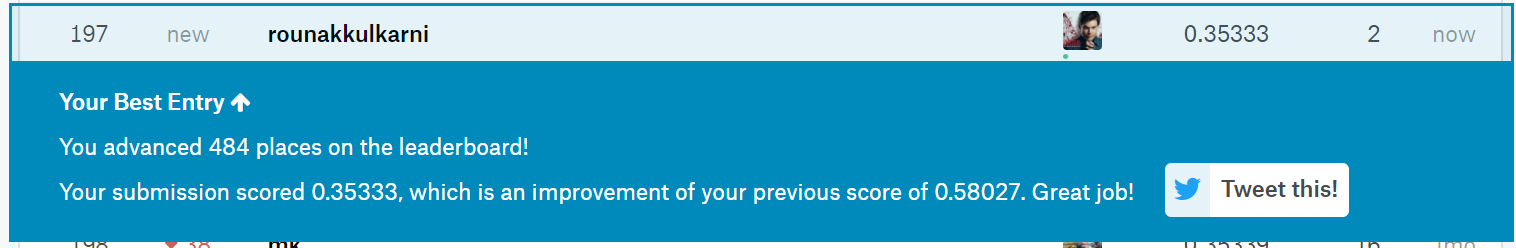
result.insert(0, 'id', X\_ID)

result.to\_csv(name+'.csv', index=False, float\_format='%.20f')

Initial submission of basic MultinomialNB model:



Second submission of MutlinomialNB model tuned using GridSearch



# **Limitations**

* Bag of Words is collection of unigrams, hence does not capture phrases and multi-word expressions, disregarding word order dependency

**Mitigation**: N-Grams (This has been used as ngram\_range=(1, 2) i.e. bi-grams in the second submission)

* Does not account for word derivations and internal structure of the sentences

**Mitigation:**

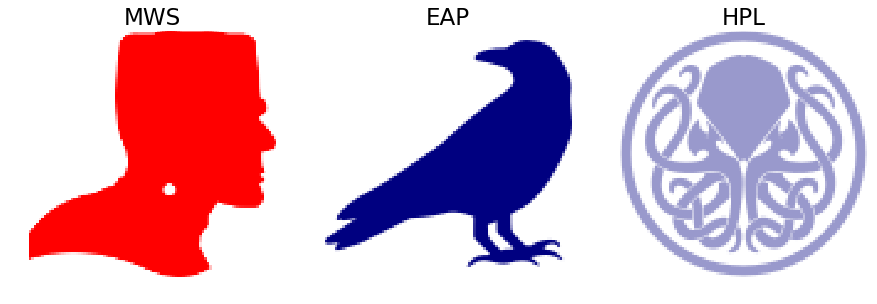
* Use token\_pattern attribute of Count Vectorizer to define good word definitions
* Use custom tokenizer that includes lemmatization

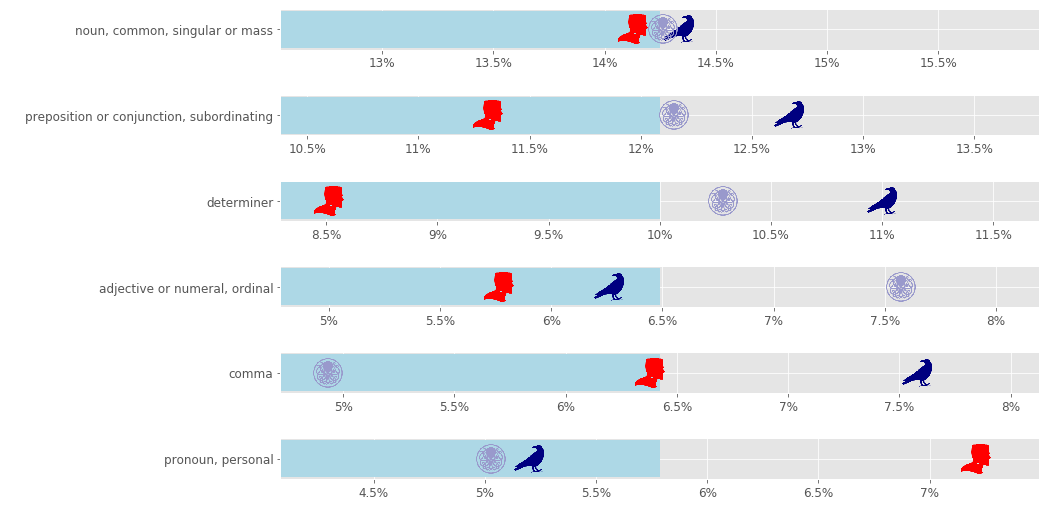
# **Exploratory Data Analysis 2**

**Notebook reference: 1\_Exploratory Data Analysis notebook**

The second phase of exploratory data analysis aims to uncover grammar insights of the sentences and find features that might help to improve the classifier model.

# **Grammar: Parts of Speech**





Above graph shows the grammar distribution for different parts of speech across authors.

# **Conclusion**

Good differences in following Parts of Speech across authors:

* + preposition or conjunction, subordinating
  + determiner
  + adjective or numeral, ordinal
  + comma

Hence, we can use parts of speech and comma usage as features to improve classification

# **Bag of Words Improvements**

3\_NLTK\_Experiments notebook

The improvements look to include the results of the Grammer: Parts of Speech analysis to improve the classifier model.

# **Custom token pattern and binary occurrence marker**

Use token\_pattern attribute of Count Vectorizer for word definitions to include comma. And use binary parameter of CountVectorizer to get better features. As noted in the feature extraction article of sckit-learn..( http://scikit-learn.org/stable/modules/feature\_extraction.html)

mnb\_clf = Pipeline([

('vect', CountVectorizer(binary=True, ngram\_range=(1,2), token\_pattern='[,]\*\w+', max\_df=0.5)),

('tfidf', TfidfTransformer(norm='l2', use\_idf=False)),

('clf', MultinomialNB(alpha=0.01)),

])

# **Customer tokenizer**

Customer tokenizer class to stem and or to lemmatize words with/without context of the sentence. The custom class can be passed to the tokenizer argument.

# Reference : http://coling.epfl.ch/TP/TP-tagging.html

def get\_wordnet\_pos(universal\_tag):

if universal\_tag == 'VERB':

return wordnet.VERB

elif universal\_tag == 'ADJ':

return wordnet.ADJ

elif universal\_tag == 'ADV':

return wordnet.ADV

else:

return wordnet.NOUN

#http://scikit-learn.org/stable/modules/feature\_extraction.html

class StemTokenizer(object):

def \_\_init\_\_(self):

self.stemmer = EnglishStemmer()

def \_\_call\_\_(self, doc):

return [self.stemmer.stem(t) for t in word\_tokenize(doc)]

class LemmaTokenizer(object):

def \_\_init\_\_(self):

self.wnl = WordNetLemmatizer()

def \_\_call\_\_(self, doc):

return [self.wnl.lemmatize(t) for t in word\_tokenize(doc)]

class LemmaTokenizer\_c(object):

def \_\_init\_\_(self):

self.wnl = WordNetLemmatizer()

def \_\_call\_\_(self, doc):

return [self.wnl.lemmatize(w[0], pos=get\_wordnet\_pos(w[1])) for w in pos\_tag(word\_tokenize(doc))]

mnb\_clf = Pipeline([

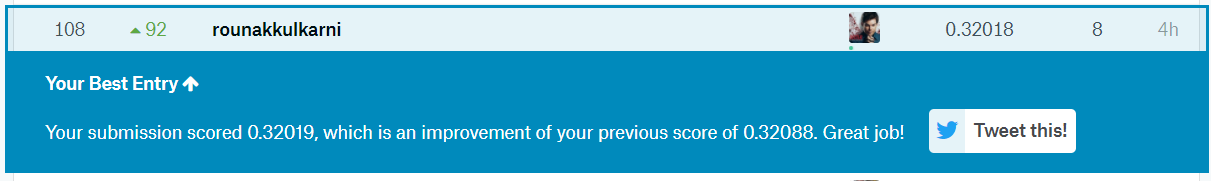
('vect', CountVectorizer(tokenizer=LemmaTokenizer()), binary=True, ngram\_range=(1,2), token\_pattern='[,]\*\w+', max\_df=0.5)),

('tfidf', TfidfTransformer(norm='l2', use\_idf=False)),

('clf', MultinomialNB(alpha=0.01)),

])

Applying the above improvement gives a slight improvement.



# **Adhoc Experiments**

# **AutoML**

Notebook reference: automl\_clf

Auto-sklearn provides out-of-the-box supervised machine learning. Built around the scikit-learn machine learning library, auto-sklearn automatically searches for the right learning algorithm for a new machine learning dataset and optimizes its hyperparameters.

automl = autosklearn.classification.AutoSklearnClassifier()

automl.fit(X\_train\_tfidf, y\_train)

y\_pred = automl.predict(X\_test\_tfidf)

y\_pred\_prob = automl.predict\_proba(X\_test\_tfidf)

print("Accuracy Score:", accuracy\_score(y\_test, y\_pred))

print("Log Loss:", log\_loss(y\_test, y\_pred\_prob))

Accuracy Score: 0.845352400409

Log Loss: 0.494008053742

# **VotingClassifier**

Notebook reference: 2\_Bag of Words

Soft Voting/Majority Rule classifier for unfitted estimators.

clf1 = SGDClassifier(alpha=1e-05, n\_iter=50, penalty='l2', loss='log')

clf2 = MultinomialNB(alpha=0.01)

eclf = VotingClassifier(estimators=[('svc', clf1), ('mnb', clf2)], voting='soft')

eclf.fit(X\_train\_transformed, y\_train)

X\_test\_transformed = pipeline\_tuned.transform(X\_test)

y\_pred = eclf.predict(X\_test\_transformed)

y\_pred\_prob = eclf.predict\_proba(X\_test\_transformed)

print("Accuracy score:", accuracy\_score(y\_test, y\_pred))

print("Log loss:", log\_loss(y\_test, y\_pred\_prob))

Accuracy score: 0.835137895812

Log loss: 0.435931677036

# **References**

* http://scikit-learn.org/stable/tutorial/text\_analytics/working\_with\_text\_data.html
* http://scikit-learn.org/stable/auto\_examples/text/document\_classification\_20newsgroups.html#sphx-glr-auto-examples-text-document-classification-20newsgroups-py
* http://scikit-learn.org/stable/modules/feature\_extraction.html#text-feature-extraction
* http://scikit-learn.org/dev/auto\_examples/model\_selection/grid\_search\_text\_feature\_extraction.html
* http://scikit-learn.org/stable/modules/feature\_extraction.html#applications-and-examples
* http://scikit-learn.org/stable/modules/ensemble.html#using-the-votingclassifier-with-gridsearch
* https://www.kaggle.com/marcospinaci/talking-plots-1-sklearn-classifiers-0-334
* https://automl.github.io/auto-sklearn/stable/
* http://scikit-learn.org/stable/auto\_examples/calibration/plot\_calibration\_multiclass.html
* http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html