Author Style Recognition

Tanishq Gupta IMT2016126
Shreyas Gupta IMT2016122
Ayush Saraswat IMT2016123

Problem Statement

Writers and literary workers have transformed the world of literary arts with their unique style of narration, character development, use of intricate vocabulary and fascinating plots. This gives us a lot of opportunity to dig through tonnes of data which can be found on the internet for analyzing what makes each author so unique.

The goal of this project is to recognize and understand different patterns and features which make up the author's unique style of writing and eventually predict who might have written a piece of work.

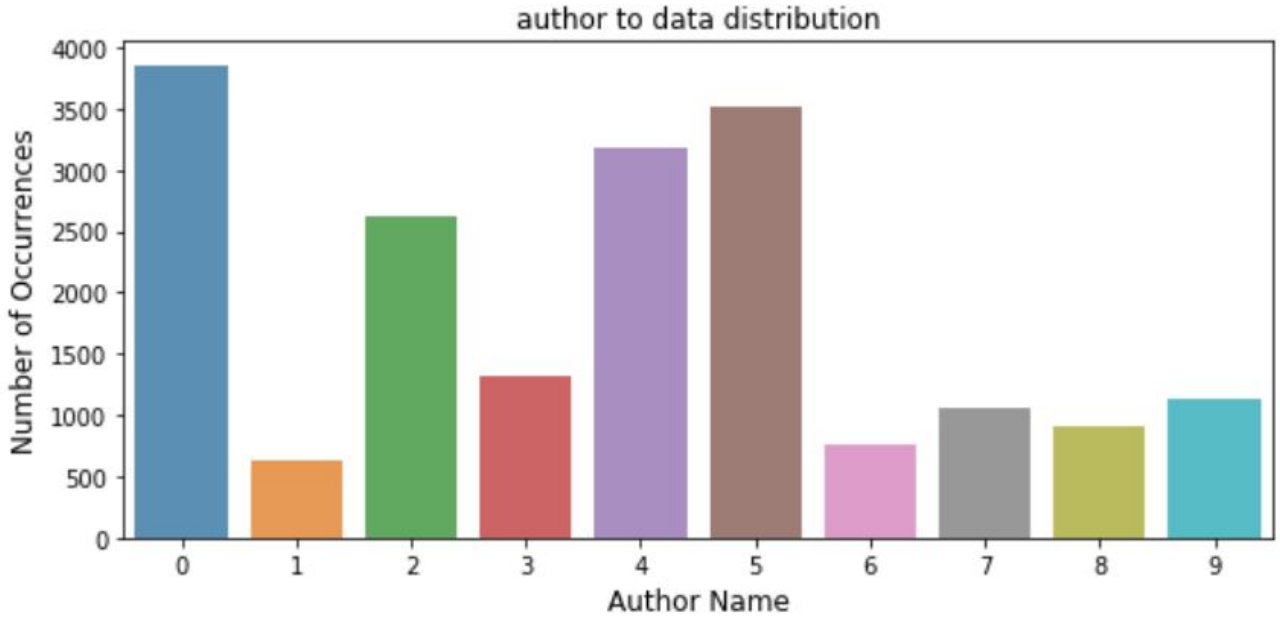
Applications

Understanding a person's unique style of writing can help literature platforms like **Wattpad** and **Goodreads** better understand their users. In a community where amateur writers share their work online, we can use vocabulary models to help emerging writers connect with people who like reading books and articles written in some particular style, and grow their follower base.

We can also use vocabulary models to create friend/follower recommendation systems by recommending connections with people who have similar likings in writing styles, eventually making the community intelligent in creating social circles and connections.

Dataset

The vocabulary model was tested and trained on a nominally small but sufficient dataset. It contains curated paragraphs from the books of 10 authors. The data was collected from Internet archive's ancient texts which were encoded in ancient encoding like ANSI and unicode. The dataset has around 16k entries shuffled with labels indicating authors. We created a test set by splitting 20% percent our train data.



Data processing

Data Cleaning

Because the data was scrapped from old encodings, this leads to a lot of mojibake (garbage data due to faulty encoding conversions) to creep into the dataset. The data contains characters which cannot be encoded in 7 bits , which leads to problems. To overcome this, we used a python library called **ftfy** which resolved most of the mojibake. The rest of the issues were manually removed with the help of regex and referring a UTF-8 encoding debugging chart which can be found here. This article by Joel Spolsky which explains the common issues people face with UTF-8 and other encodings was really helpful!

code to clean data

```
train.text = train.text.apply(lambda x:
re.sub(r'\"', r"""", re.sub("†", "", ftfy.fix_encoding(re.sub("", " ",
re.sub(" {2,}", " ", re.sub("\([0-9]*\)|\[[0-9]*\]", " ", re.sub("\r|\n", " ", x))))))))
```

Feature Extraction

Once the data is cleaned, we extract and create 508 features which include

tf-idf features -
or term frequency - inverse document frequency features , is a method to evaluate how important is a word in a document. It converts textual representation into Vector Space Model.

We did not stem the data because the tense of the word plays quite an important role in understanding the writing style.

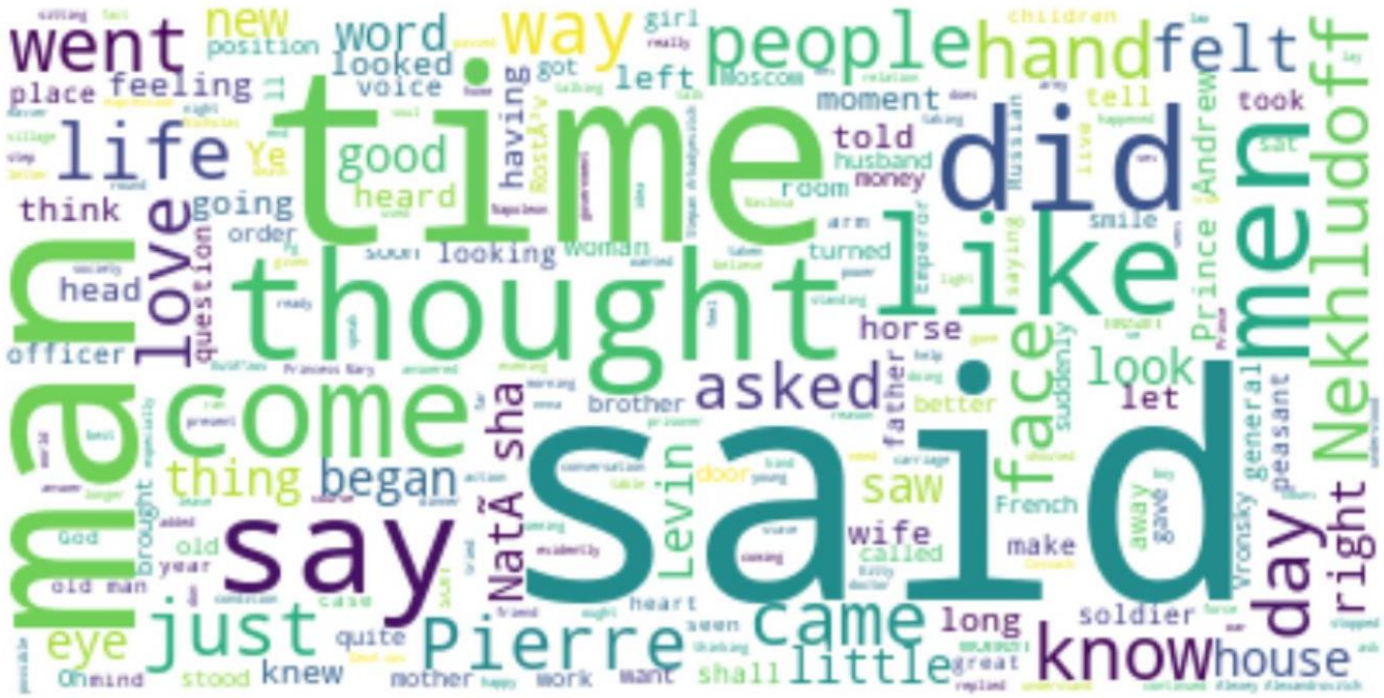
We have used ngrams_range (1, 4). We have done this because having a combination of more than 2 words will give us more intrinsic details about the writing style of the writer. We tested with various ngrams_ranges and this range seems the most optimal.

Also with so many ngrams we face the problem of having too many features. Thus we have used **Truncated SVD** (aka Latent Semantic Analysis) which reduced the dimensionality of the tf-idf features to 500 features from approximately 100 thousand features.

```
#tf-idf feature creation with 1-4 ngram range
tfidf_vec = TfidfVectorizer(
    max_df = 0.3,
    min_df = 3,
    lowercase = True,
    stop_words = eng_stopwords,
    ngram_range = (1,4),
    analyzer = 'char'
```

```
)
train_tfidf = tfidf_vec.fit_transform(train.text)
indices = pd.DataFrame(tfidf_vec.get_feature_names())


# svd-based feature dimension reduction
n_comp = 500
svd_obj = TruncatedSVD(n_components = n_comp, algorithm = 'arpack')
svd_obj.fit(train_tfidf)
train_svd = pd.DataFrame(svd_obj.transform(train_tfidf))
train_svd.columns = ['svd_char_' + str(i) for i in range(n_comp)]
train = pd.concat([train, train_svd], axis=1)
del train_tfidf, train_svd
```

Unigrams Word Cloud



PointOfView

Writers can be differentiated with the point of view in which they usually write their novels. Novels written in third person usually have a lot of quotations. The feature is created by finding the frequency of quotation marks in some text.

```
train['perspective'] = train.text.apply(lambda x: len(re.findall("\“",x)))
```

Informality Score*

Amateur writers and even professionals sometimes lean towards a more informal tone. We can notice this if there are a lot of “.....” or a lot of informal words. Because we aren’t allowed to use pre-trained models to get informality scores from external libraries, we won’t use them.

```
train['informality'] = train.text.apply(lambda x:
len(re.findall("(\. ){2,}|(\.){2,}", x)))
```

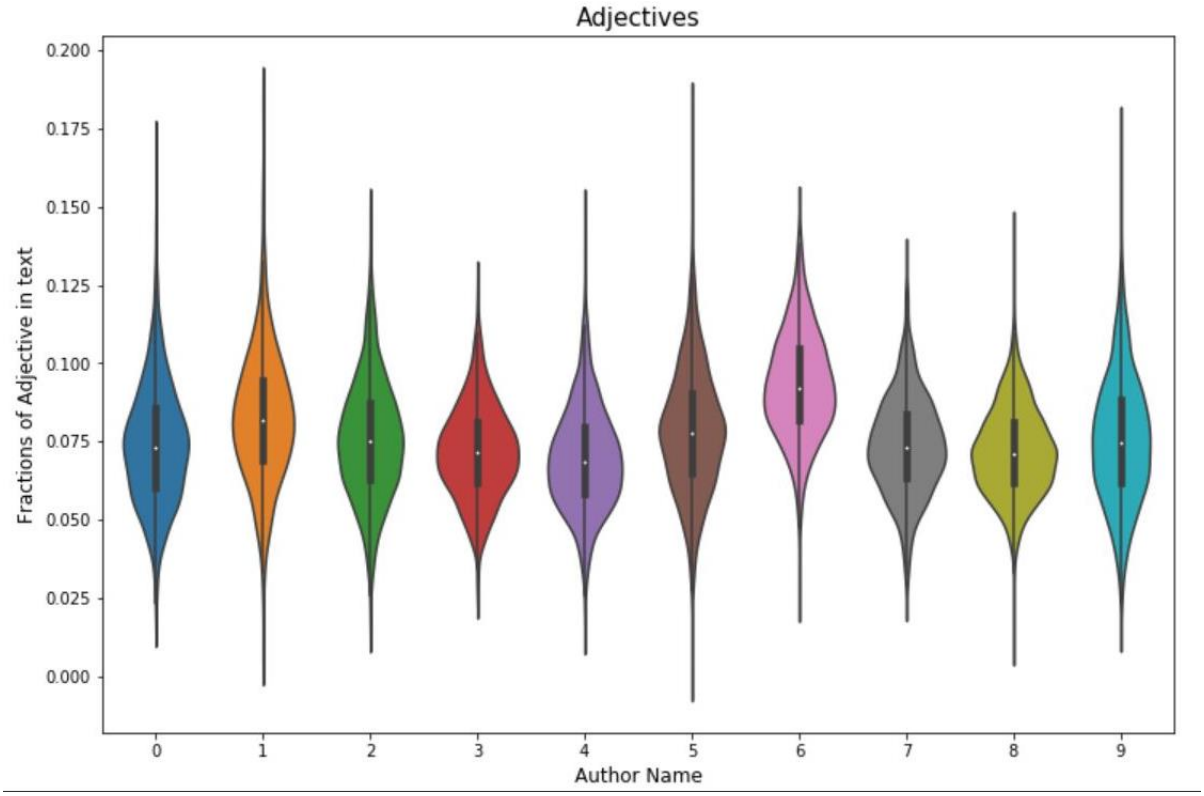
Meta Features

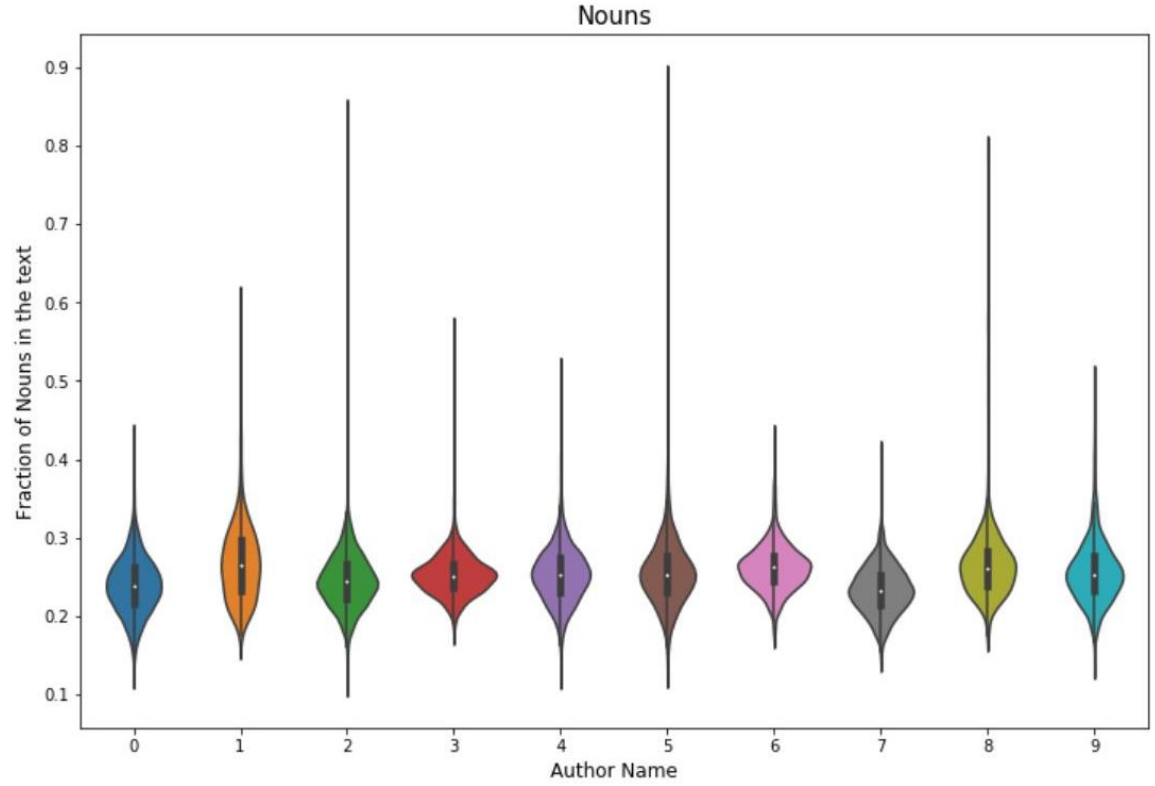
This helps us get a better picture of the author’s vocabulary and will allow us to capture the sense of his writing. The feature list is... * Fraction of unique words * Fraction of stop words * Fraction of punctuations * Fraction of Nouns * Fraction of Verbs * Fraction of Adjectives

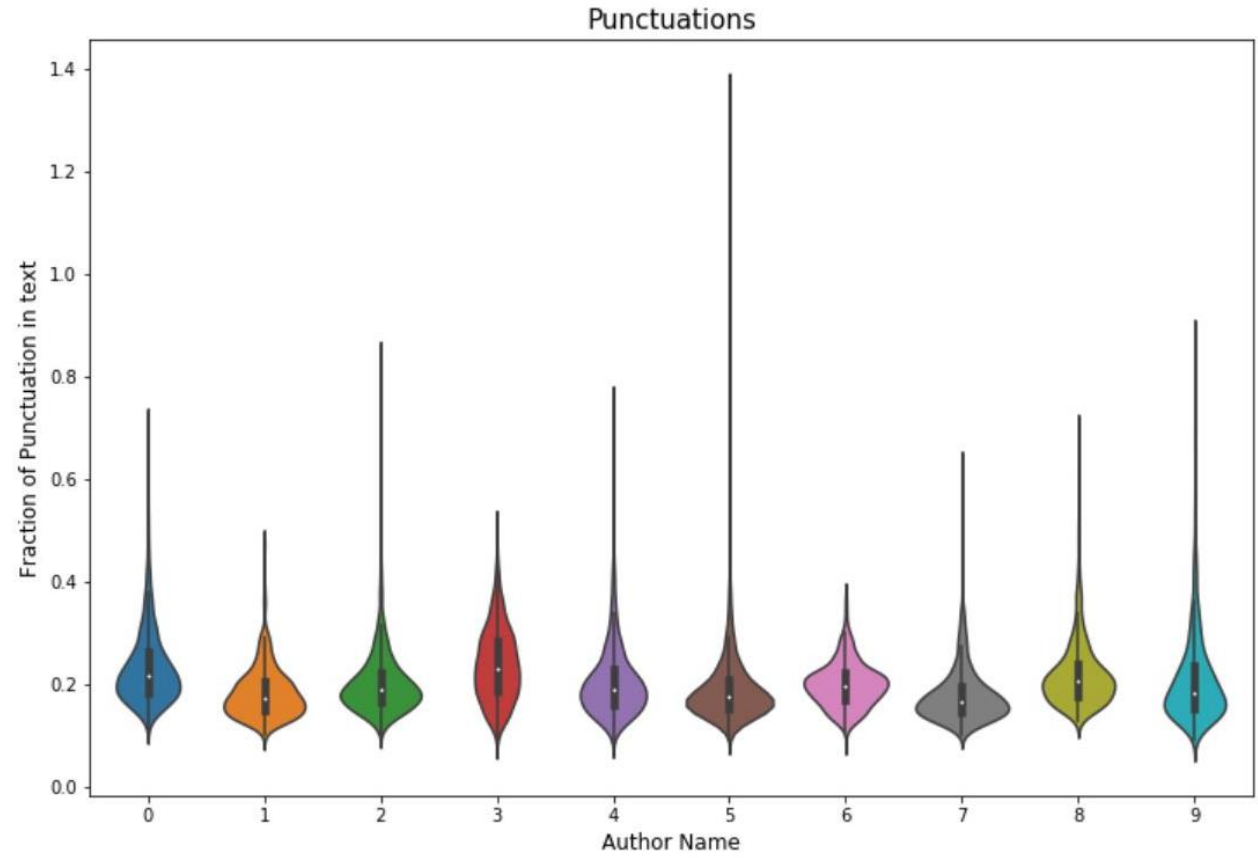
```
#total no of words in the dataset
num_words = train.text.apply(lambda x : len(str(x).split()))

#calculation of fractions of new feature columns
train['fraction_unique_words'] = train.text.apply(lambda x : len(set(str(x).split()))) / num_words
train['fraction_stopwords'] = train.text.apply(lambda x: len([l for l in str(x).lower().split() if l in
eng_stopwords])) / num_words
train['fraction_punctuations'] = train.text.apply(lambda x: len([ch for ch in str(x) if ch in
string.punctuation])) / num_words
train['fraction_nouns'] = train.text.apply(lambda x: len([poc for poc in nltk.pos_tag(str(x).split()) if poc[1]
in ('NN','NNP','NNPS','NNS')])) / num_words
train['fraction_adj'] = train.text.apply(lambda x: len([poc for poc in nltk.pos_tag(str(x).split()) if poc[1] in
('JJ','JJR','JJS')])) / num_words
```

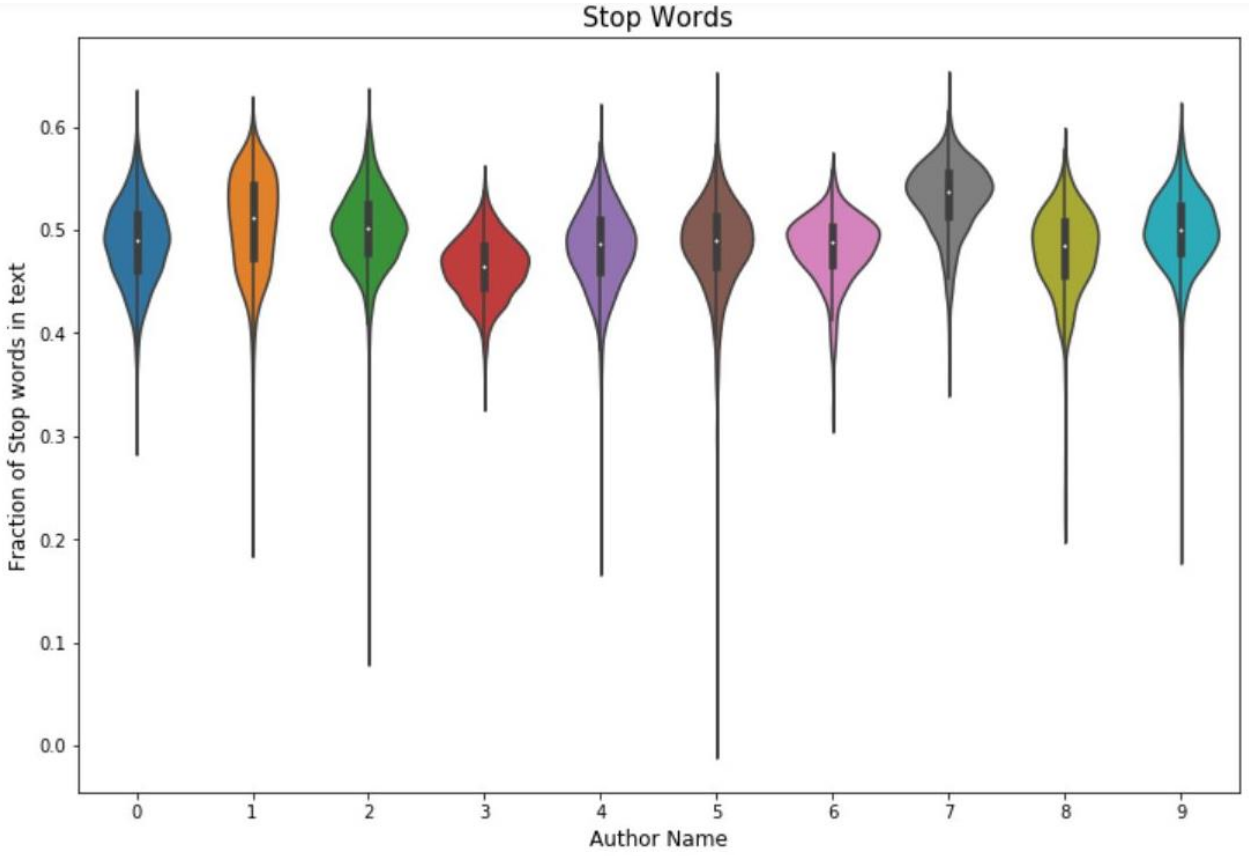
```
train['fraction_verbs'] = train.text.apply(lambda x: len([poc for poc in nltk.pos_tag(str(x).split()) if poc[1] in ('VB','VBD','VBG','VBN','VBP','VBZ')])) / num_words
```

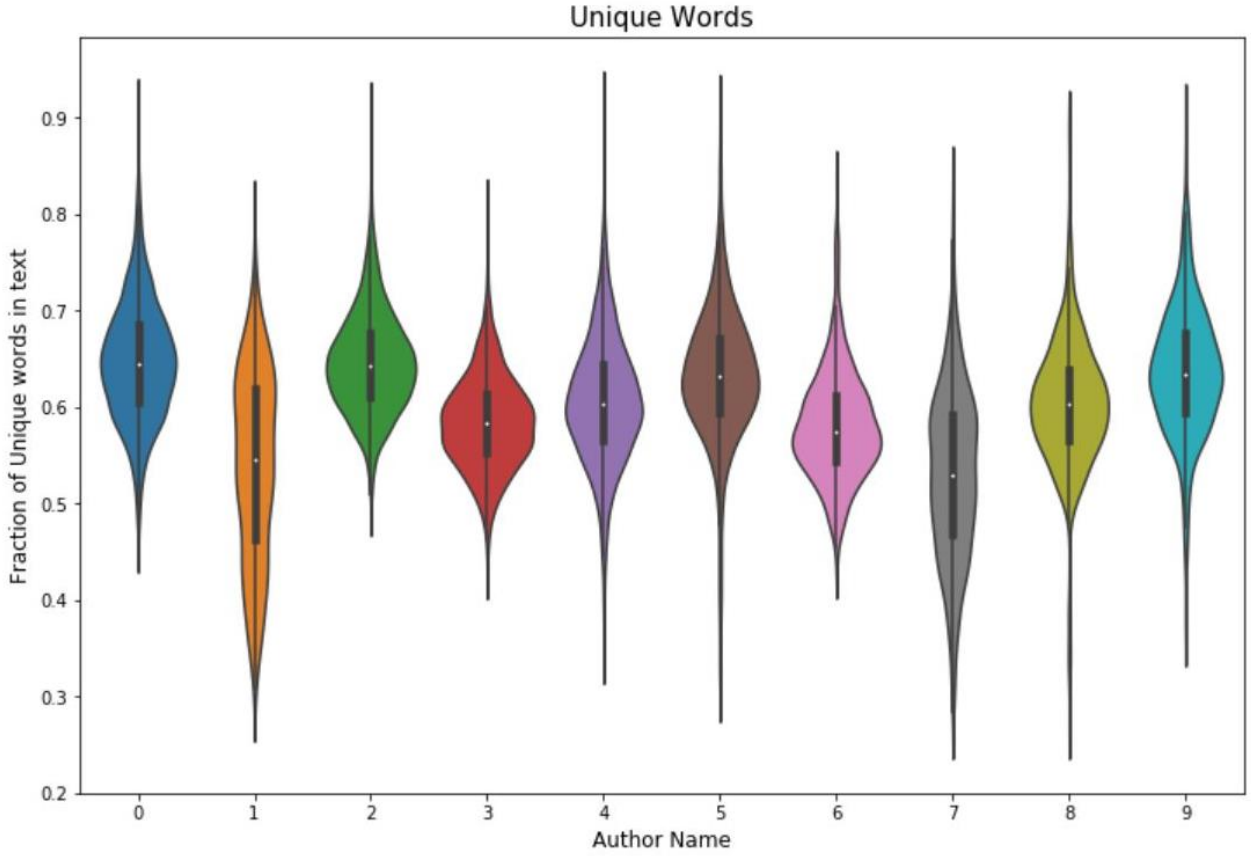
We have visualized the above features using violin plots and have removed outliers accordingly.

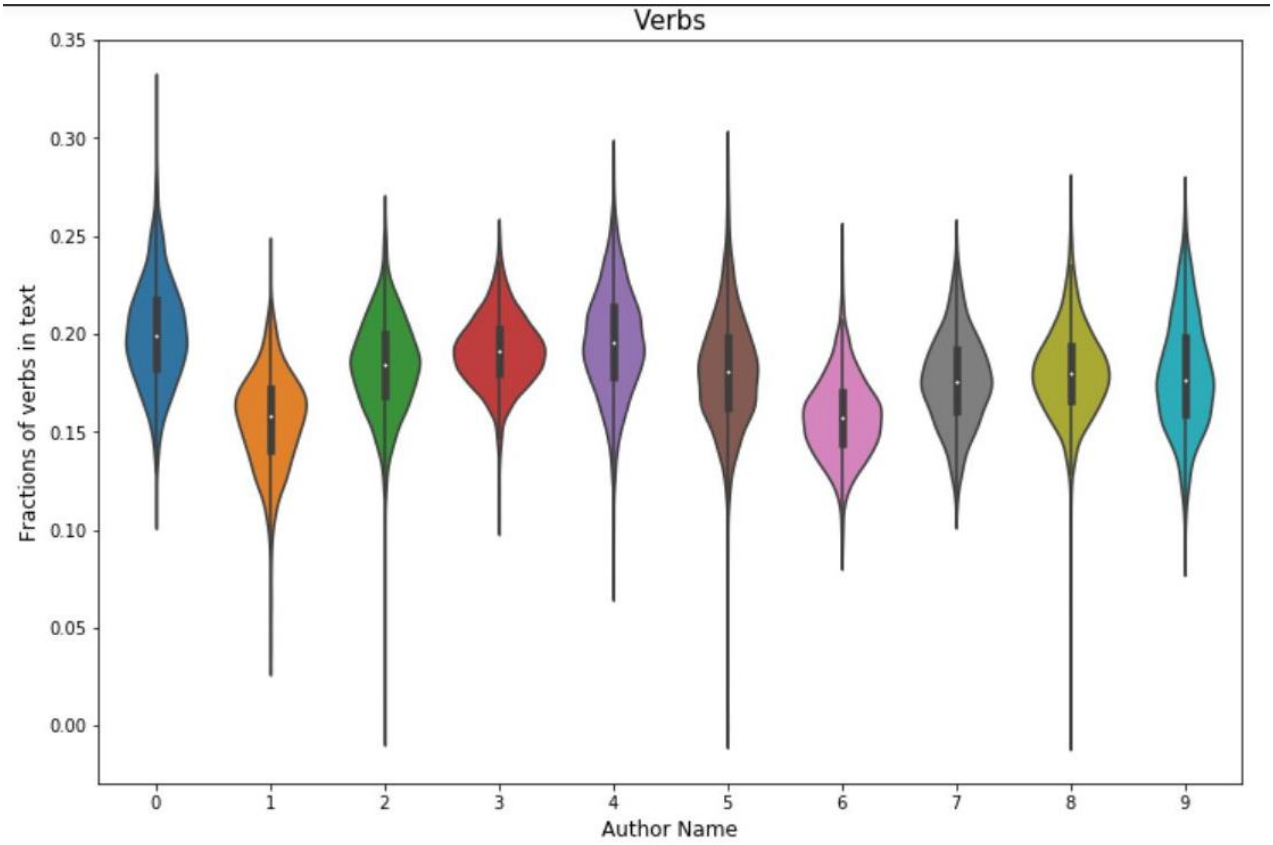



Nouns
Fraction of Nouns in the text
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6
7
8
9
Author Name


Punctuations
Fraction of Punctuation in text
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
9
Author Name


Stop Words
Fraction of Stop words in text
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
1
2
3
4
5
6
7
8
9
Author Name


Unique Words
Fraction of Unique words in text
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
1
2
3
4
5
6
7
8
9
Author Name



Among all the approaches we took, we also tried to add sentiment analysis to the text data. But soon realized sentiment analysis doesn't make sense as a writer can have different sentiment in his different writings from happy to sad to horror and to whatnot.

Training

We have used 2 models Logistic Regression and SGD Classifier. Using Logistic Regression we got the accuracy score of **95.2 %** . Using Multinomial Naive Byes we got the accuracy of **92.2%**.

Future Goals

We can improve the training process by using LSTMs. We can also use Glove and other pre trained models like Word2Vec to further understand the semantic and lexical features of the dataset.

We can possibly use Markov Models which will remember previous words and emit probability distribution over the next words in the sequence, and we can use this to create probability distributions of the author given a sentence i.e. P(Author | Sentence).