Classification of proetry on the basis of different genre

In this project the poetry has been classified into different geners using two model namely 1) Naive bayes classifier 2) Logistic Regresson for classification

The data set has been taken form Kaggle (https://www.kaggle.com/ultrajack/modern-renaissance-poetry (https://www.kaggle.com/ultrajack/modern-renaissance-poetry))

In [ ]:

```
import nltk
from nltk.corpus import stopwords
set(stopwords.words('english'))
```

In [115]:

```
import numpy as np
import sklearn as sk
import pandas as pd
dataset=pd.read_csv("C:/Users/Nirvan Dogra/Desktop/poems-from-poetryfoundation-org/all.
csv")
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
print(dataset.describe())
X_train, X_test, y_train, y_test = train_test_split(dataset['content'], dataset['type']
, test_size=0.2, random_state=42) #splitting the data into test and training
```

```
                     author                                     content  \
count                   573                                         573
unique                   67                                         506
top     WILLIAM SHAKESPEARE  Originally published in Poetry, March 1914.
freq                     71                                           4

       poem name          age  type
count        571          573   573
unique       508            2     3
top      Canto IV  Renaissance  Love
freq           3          315   326
```

About Data

-> The data set contains 573 data values -> It has 5 colums namely (author, content, poem name, age, type) -> The most common vaue is William Shakespeare containg a totat of 71 data values

Pre-pocessing of data

For the pre-porcessing of data a library named nlkt has been used.For the pre-processing of data the following steps have been take: 1) Conversion to lower case 2) Bag of words 3) Removal of stop words 4) Removal of puntuation 5) Lemmatizer 6) conversion of text to sparse matrix

In [116]:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer


def text_process(example_sent):
    example_sent = example_sent.lower()
    stop_words = set(stopwords.words('english'))

    word_tokens = word_tokenize(example_sent)

    punctuation=['.',',','!',';','(',')','-', '_']
    filtered_sentence = []
    for w in word_tokens:
        if not (w in stop_words) and not (w in punctuation):
            filtered_sentence.append(w)
    return(filtered_sentence)


#     lemmatizer = WordNetLemmatizer()
#     base_form=[];
#     for w in filtered_sentence:
#         print(w," ", lemmatizer.lemmatize(w, pos='a'))
#         base_form.append(lemmatizer.lemmatize(w, pos='a'))
#     print(base_form);

# example_sent = "This is a sample sentence, showing off the stop words filtration."
# for example_sent in X_train:

#     example_sent = example_sent.lower()
#     stop_words = set(stopwords.words('english'))

#     word_tokens = word_tokenize(example_sent)
#     print(word_tokens)
#     punctuation=['.',',','!',';','(',')','-', '_']
#     filtered_sentence = []
#     for w in word_tokens:
#         if not (w in stop_words) and not (w in punctuation):
#             filtered_sentence.append(w)
#     print(filtered_sentence)


#     lemmatizer = WordNetLemmatizer()
#     base_form=[];
#     for w in filtered_sentence:
#         print(w," ", lemmatizer.lemmatize(w, pos='a'))
#         base_form.append(lemmatizer.lemmatize(w, pos='a'))
#     print(base_form);

#     word2count={}
#     for w in base_form:
#         if w not in word2count.keys():
#             word2count[w] = 1
#         else:
#             word2count[w] += 1

#         print(word2count);
```

```
#                  # Importing necessary libraries

#          # instantiating the model with Multinomial Naive Bayes..
#          model = MultinomialNB()
#          # training the model...
#          model = model.fit(base_form, y_train)

# filtered_sentence = [w for w in word_tokens if not w in stop_words]

#

# for w in word_tokens:
#     if w not in stop_words:
#         filtered_sentence.append(w)

# print(word_tokens)
# print(filtered_sentence)
```

In [117]:

```
bow_transformer=CountVectorizer(analyzer=text_process).fit(X_train)
# transforming into Bag-of-Words and hence textual data to numeric..
text_bow_train=bow_transformer.transform(X_train)
# transforming into Bag-of-Words and hence textual data to numeric..
text_bow_test=bow_transformer.transform(X_test)
```

Naive Bayes #

In [111]:

```
from sklearn.naive_bayes import MultinomialNB
# instantiating the model with Multinomial Naive Bayes..
model = MultinomialNB()
# training the model...
model = model.fit(text_bow_train, y_train)
```

In [73]:

```
model.score(text_bow_train, y_train)
```

Out[73]:

```
0.868995633187773
```

In [74]:

```
# Importing necessary libraries
from sklearn.metrics import classification_report

# getting the predictions of the Validation Set...
predictions = model.predict(text_bow_test)
# getting the Precision, Recall, F1-Score
print(classification_report(y_test,predictions))
```

```
                     precision    recall  f1-score   support

               Love       0.72      0.87      0.79        68
Mythology & Folklore       0.20      0.11      0.14         9
             Nature       0.68      0.50      0.58        38

          micro avg       0.69      0.69      0.69       115
          macro avg       0.53      0.49      0.50       115
       weighted avg       0.67      0.69      0.67       115
```

Inferring form the results

The model has an accuracy of 86.89% The model works best at classifying 'Love' and worst for the classification of 'Mythology & Folklore'. To compensate for this, we can use weighted metrics on the Mythology and Folklore classes.

Logistic regression #

The preprocessing of the data remains the same.

In [113]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
lm=LogisticRegression(solver='lbfgs', multi_class='multinomial').fit(text_bow_train, y_
train)
predicted_classes = lm.predict(text_bow_test)
#print(predicted_classes)
accuracy = accuracy_score(y_test,predicted_classes)
parameters = model.coef_
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:758: ConvergenceWarning: lbfgs failed to converge. Increase the number o
f iterations.
  "of iterations.", ConvergenceWarning)
```

Explanation of non convergence:

The model doesnt converge due to the presence of mulitple local minima, each sub-localized to the genre. Since there are multiple genre to be covered, it only makes sense that the minima is different for each genre.

In [114]:

```
print(accuracy)
```

```
0.6782608695652174
```

Inferring from the results

The accuracy of the model is 67.82% This model preforms worse than the previous one.

Final comments

The accuracy of the above model can be improved with the use of explicit minima, i.e. using different minima in training models for each genre. The accuracy of the earlier model can be improved with a change in either pruning via AdaBoost, or by using weighted metrics and post pruning the model.