ECSE 551 Group 5 Mini-Project 2

Sara Yabesi
McGill University
sara.yabesi@mail.mcgill.ca

Baharan Nouriinanloo
McGill University
baharan.nouriinanloo@mail.mcgill.ca

Mahta Amini
McGill University
mahta.amini@mail.mcgill.ca

Abstract

This study examines the effectiveness of various machine learning classifiers for text classification, focusing on the Naïve Bayes and Multi-layer Perceptron classifiers (MLP). The dataset consists of posts and comments from Reddit that have been classified into four possible categories, including Trump, Obama, Musk and Ford. Model selection was performed using 5-Fold Cross-Validation, and the results showed that the accuracy of the final model on the test set depends on various techniques, such as the preprocessing techniques and the choice of the classifier. The GridSearchCV method was utilized to find the optimal combination of preprocessing pipeline and classifier parameters. Bernoulli Naïve Bayes is commonly used for text classification, and we have implemented this classifier from scratch. We have also utilized multiple classifiers from Scikit-Learn, such as Support Vector Machine, Logistic Regression, Multinomial Naive Bayes, Decision Tree and Random Forest. In addition, we have performed exhaustive hyper-parameter tuning for all of these classifiers and our Bernoulli Naïve Bayes classifier. Various other techniques have also been used to build our models, such as Lemmatization, Stemming, utilizing a stop-word dictionary, Feature Selection(chi-squared test) and Normalization. Another technique that also proved to be powerful was Ensemble learning. Combining multiple classifiers together proved to yield high accuracy. However, the best accuracy was produced by Multi-layer Perceptron classifiers combined with normalization, utilizing a stop-word dictionary and a Chi-squared test.

1 Introduction

Text analysis [1] is the process of extracting valuable insights and meaning from written or spoken words using computational techniques. It aims to identify patterns and relationships in a large volume of text data. Text analysis has become an essential tool for businesses and researchers who deal with large amounts of textual data and need to extract meaningful information from it. In this project, we aim to apply text analysis to data from the Reddit website. We have a Train dataset that contains 718 posts and comments from four different subreddits: Trump, Obama, Musk, and Ford. We also have a Test dataset. The objective is to determine the subreddit where the comment was originally published in the test dataset. By doing this project, we assess the performance of various text classification algorithms and evaluate the performance of various classifiers. The study has also explored data preprocessing and feature selection techniques, directly impacting the text classifier performance. Data preprocessing techniques include removing punctuation, removing numeric values, lowercasing the words, lemmatization [2], stemming [3] and using different dictionaries as stop-words. We have also examined different techniques for feature selection, such as the chi-squared test [4] and mutual information [5].

During the experiments, aside from Bernoulli Naïve Bayes (NB) [6] and Multinomial Naïve Bayes classifiers, various other classifiers [7] were used for prediction. These classifiers were provided by

the Scikit-Learn [8] library and included Linear Support Vector Machine, Multi-layer Perception [9], Logistic Regression, Decision Tree and Random Forest. We have also stacked multiple classifiers together [10]. Stacking is a powerful technique that can lead to improvement in the performance of the model. To compare the classifiers' accuracy, an optimal data preprocessing pipeline was employed for all of them, and the optimal pipeline hyper-parameters were determined using Scikit-Learn's GridSearchCV function [11]. The classifiers' accuracy was measured using their respective 5-Fold Cross-Validation scores. The highest accuracy on the test set belonged to the MLP classifier, with 97.59%.

2 Dataset

2.1 Overview

It is crucial to properly analyze and comprehend the characteristics of the dataset prior to conducting any investigation on machine learning models. Machine learning algorithms are utilized to solve a multiclass text classification problem. The training dataset consists of 718 comments that belong to one of four possible subreddits or classes: "Trump," "Obama," "Musk," and "Ford." The subreddit labels are almost uniformly distributed, with 179 comments from the Ford and Musk subreddit and 180 from the Obama and Trump subreddit. On average, these comments contain 102 words with a minimum of 1 word and a maximum of 557 words and are ordered by the subreddit they belong to. The test set consists of 279 comments that our algorithm must assign to one of the aforementioned subreddits. On average, in the test set, comments contain 103 words with a minimum of 1 word and a maximum of 496 words.

2.2 Feature Extraction

For text classification to be feasible, extracting significant numerical features from the raw text data is essential. The comments in both the training and test datasets must undergo several preprocessing stages to transform them into feature vectors. To achieve this goal, we perform several preprocessing steps and investigate whether they improve our model's accuracy.

- **Removing Punctuation:** Removing punctuation is a preprocessing step in natural language processing and machine learning. Punctuation marks carry little meaning on their own and can add unnecessary noise to the text data, which can affect the accuracy of the analysis.
- **Lowercasing:** This technique converts all text to lowercase, which helps to reduce the dimensionality of the data and treat different cases of the same word as the same feature, thereby improving the accuracy of the analysis.
- **Removing Numerical Values:** Depending on the classification problem, numerical values may add little value in text data analysis, and their presence can add noise to the text data. Therefore, removing numerical values can help improve the analysis's accuracy. While constructing our models, we test whether performing a combination of these steps improves the model's accuracy.

2.3 Shuffling

It is important to shuffle the dataset to prevent the model from learning any implicit patterns based on the ordering of the labels in the training dataset. Shuffling ensures that the model learns to generalize and make accurate predictions regardless of the order in which the data is presented. In our case, the training data is initially ordered by their subreddit label.

2.4 Stop Words

Stop words refer to commonly occurring words in all documents, such as articles, prepositions, pronouns, and conjunctions, which are the most frequently used words in any language. These words do not contribute significantly to the meaning or information conveyed by the text. We employed three distinct categories of stop words: NLTK's [12] 179-element stop word list, scikit-learn's 318-element stop word list, and our custom list containing 590 elements. Our custom dictionary was built based on inspecting the training dataset. We investigate the performance of different unions of these dictionaries on our machine-learning model.

2.5 Lemmatization and Stemming

Both lemmatization and stemming are techniques used to decrease the number of features in a text by grouping different forms of a word. While stemming simply tries to reduce a word to its root

form without considering the context, lemmatization is a more accurate approach. Lemmatization considers the word's context and retains the root word's readable form. Therefore, lemmatization generally yields better results than stemming. However, stemming is a more straightforward process than lemmatization; it can often be faster and require fewer computational resources. The number of words these methods remove may differ slightly depending on the analyzed text.

2.6 N-gram

N-gram [13] is a widely used method in NLP that captures the word sequence in a text by extracting n consecutive items from it to create features. Although incorporating these features can improve the accuracy of predictions, it may also lead to a significant increase in computational costs due to the resulting expansion in feature size. While building our model, we explore different ranges of n-values such as (1,1),(1,2) and (1,3).

2.7 Normalization

The technique of normalizing data to z-scores is widely used in machine learning to centralize the value distribution of the data without compromising accuracy [14]. However, this approach may not produce desirable outcomes when applied to text-based datasets like Reddit because the numerical representation of features typically denotes the frequency of word occurrence in each comment, resulting in values that are either 0 or 1, thus normalization may not be effective.

2.8 Vectorization

In natural language processing, a token is any character sequence separated by a separator such as white space or punctuation. The bag of words approach is a common vectorization technique used to represent text documents as a set of all the words they contain. Although computationally efficient, this method has limitations, such as the inability to capture token position and co-occurrences in different documents [14]. Count vectorization is the simplest form of vectorization, while TF-IDF(Term Frequency-Inverse Document Frequency) addresses issues of frequently occurring words by assigning weights based on term frequency and inverse document frequency.

3 Proposed Approach

This section will delve into the details of the classifier we have implemented from the ground up, as well as the classifiers we have employed from the scikit-learn library. Furthermore, we will elaborate on the rigorous methodology we employed for model optimization and hyperparameter tuning, along with various feature selection techniques that were carefully considered.

3.1 Feature Selection

When working with a large corpus of text in text classification, it is common to encounter many features that offer little predictive value. Eliminating these irrelevant features can improve accuracy while reducing the computational cost of fitting the model. Several metrics can be used to determine the importance of a feature, such as information gain, F-value regression score, chi-squared statistic, and ANOVA (analysis of variance) f-test statistic [15]. We test different feature selection techniques and choose chi-squared statistics as the final feature selection technique. It is easy to compute and does not significantly increase the time required to fit the models. Although mutual information gain is a preferred method, it was not feasible due to long computation times. We investigate different k values for the number of features to be selected.

3.2 Classifier Selection

- **Bernoulli Naive Bayes:** Bernoulli Naive Bayes, a member of the Naive Bayes family, leverages the Bernoulli Distribution, which restricts the feature values to binary form, exclusively 0 or 1. Therefore, it is best suited for datasets containing binary features [7]. We employ the Bernoulli Naive Bayes algorithm in our model development, given its aptness for our dataset's feature representation. We also employ Laplace smoothing parameter for our hyperparameter tuning.
- **Multinomial Naive Bayes:** Multinomial Naive Bayes is a valuable variation of the Naive Bayes algorithm in machine learning, especially when working with a multinomial distribution dataset. It distinguishes between the frequency of a term in a feature vector, while Bernoulli Naive Bayes only models the occurrence or non-occurrence of a feature using binary encoding. In our implementation, we further refine our model through hyperparameter tuning, utilizing the Laplace smoothing parameter.

- **Support vector machine:** (SVM) algorithm is a binary classifier that builds an optimal hyperplane through a linear kernel in the feature space. SVM demonstrates its prowess in high-dimensional spaces, and its performance can be affected by the selection of kernels. By constructing a hyperplane that maximizes the margin from the closest data points in any class, SVM effectively separates data, resulting in a lower generalization error for the classifier.The hyperparameters we employ contain the Regularization parameter(C), kernel(linear) choice and gamma [7].
- **Logistic Regression:** Logistic regression is a classification algorithm that assigns observations to a discrete set of classes based on probability. While Logistic Regression can be considered a Linear Regression model, it uses a more complex cost function known as the Sigmoid function or logistic function instead of a linear function. The hyperparameters we employ contain the Regularization parameter(C), the choice of a penalty function and the maximum number of iterations.
- **Decision tree:** Decision tree (DT) is a widely used non-parametric supervised learning technique for classification and regression tasks. The DT algorithm sorts down the tree from the root to the leaf nodes to classify instances. Instances are classified by checking the attribute defined by that node, starting at the tree's root node and then moving down the tree branch corresponding to the attribute value. For hyperparameter tuning, we explore different values for criterion, maximum depth of the tree, maximum number of features and minimum number of samples required to split.
- **Random forest:** Random Forest is a widely-used supervised machine learning algorithm that builds decision trees on different samples and takes their majority vote for classification and the average for regression. It uses parallel ensembling to fit several decision tree classifiers in parallel, minimizing over-fitting and increasing prediction accuracy. It is adaptable to classification and regression problems and fits both categorical and continuous values well. We have the same set of hyperparameters as Decision Tree for tuning the model and also the number of trees in the forest.
- **MLP Classifier:** The multilayer perceptron (MLP) is a neural network architecture that employs a feedforward topology comprising an input layer, one or more hidden layers, and an output layer. In order to optimize the model, backpropagation is utilized for adjusting the weights. Additionally, the MLP is characterized by various hyperparameters, including the number of hidden layers, number of neurons in each layer, and number of iterations. Furthermore, the connectivity of each layer is fully linked to the following layer, which can incorporate a nonlinear activation function.
- **Ensemble learning** Ensemble learning is a machine learning technique that involves combining multiple models to improve predictive performance. Ensemble methods can reduce overfitting and improve generalization by combining the strengths of different models. Examples of ensemble methods include bagging, boosting, and stacking, which combine multiple models to achieve better results. Ensemble learning has been successfully applied in various domains, including computer vision, natural language processing, and predictive analytics. We have implemented two types of ensemble learning as follows: First, combining Multinomial Naive Bayes and Logistic Regression. The second combines Multinomial Naive Bayes and MLP Classifier.

3.3 Pipeline and Hyper-parameter Tuning

A pipeline approach can be employed to identify the optimal feature and model combinations. This entails combining feature engineering, pre-processing, and modelling steps into a unified object called a pipeline. The ideal model can be determined by leveraging GridSearchCV, which explores diverse classifier hyperparameters and selects optimal settings. Using pipelines makes the process more streamlined and reduces the possibility of errors. Using GridSearchCV, numerous assumptions and hyperparameters can be tested, enabling the identification of the optimal combination that generates the best outcome.

4 Results

We build our models using seven different classifiers as mentioned in section 3.2 and two ensemble learners. These models have been exhaustively trained using GridSearchCV to arrive at the best set of parameters. For each model, we tune the corresponding hyperparameters. Each step in 2.2 is implemented, and the accuracy is recorded to see which technique performs best. In none of our methods, lemmatization or stemming improves the final accuracy. However, in the ensemble learners, removing numeric values and lowercasing improves the model's accuracy. Normalization almost always proved to increase the final accuracy. Normalization did not improve the results for two out of 9 models(Logistic regression and Decision tree). The implemented model's complete list and configurations are demonstrated in table 1. Our final model is build with MLP classifier with

activation=relu,hiddenlayersizes=100, maxiteration=2000,solver=adam, alpha=0,1. We have used a normalizer, a TfIdfVectorizer, and our own custom dictionary for stop word removal and have chosen 2500 features using the chi-squared test. The model has a 94.983% accuracy in k-fold validation with *k=5* and 97.59% on our test dataset in Kaggle. Regarding accuracy, the second model was our ensemble learner that combined MLPClassifier and Multinomial Naïve Bayes. In this model, removing numerical values and lowercasing the words as preprocessing steps lead to an increase in the model's performance. For feature selection, we have selected *k=5000* features. We have also utilized a normalizer.

	Model Setting	5-Fold Avg. Accuracy	Test Set Accuracy
1	SVM(gamma=0.1,kernel=linear)+Stop words(NS*)+Normalizer+CountVec+Chi2(3000)	90.528%	90.361%
2	CustomBernoulliNB(alpha=0.5)+Stop words(Custom**)+Normalizer+TfIdfVec+Chi2(5000)	94.290%	93.975%
3	MultinomialNB(alpha=0.5)+Stop words(Custom)+CountVec+Chi2(5000)	94.011%	93.975%
4	LogisticRegression(c=10,solver=sag,maxiteration=1000,penalty=l2,classweight=balanced) +Stop words(NS)+CountVec+Chi2(5000)	93.037%	89.156%
5	DecisionTree(criterion=gini,maxdepth=100,minsampleleaf=1,minsamplesplit=5) +Stop words(Custom)+CountVec+Chi2(5000)	88.444%	91.566%
6	Random Forest(nestimator=100)+Stop words(NS)+Normalizer(norm=max) +CountVec(Binary=True)+Chi2(3000)	89.274%	92.771%
7	MLPClassifier(activation=relu,hiddenlayersizes=100, maxiteration=2000,solver=adam, alpha=0,1)+Normalizer+Stop words(Custom)+TfIdfVec(max_df: 0.25)+Chi2(2500)	94.983%	97.59%
8	LogisticRegression(solver=sag)+MultinomialNB(alpha=0.5)+Normalizer +TfIdfVec(max_df: 0.75)+Stop words(Custom)+Chi2(5000)+removeNumeric+Lowercase	95.123%	95.18%
9	MLPClassifier(solver=lbfgs,hiddenlayersize=32,alpha=0.1)+MultinomialNB(alpha=0.1)+ Normalizer+TfIdfVec(max_df:1)+Stop words(Custom)+Chi2(5000) +removeNumeric+Lowercase	95.402%	96.385%

Table 1: Model details and their k-fold accuracy and test accuracy.
NS* is the union of NLTK & scikit-learn stop word dictioinary.Custom** is our custom dictionary.

5 Discussion and Conclusion

The study's findings indicate that the Multi-layer Perceptron classifier using the chi-squared technique as a feature selection provides the most accurate result. It has an accuracy of 97.59% on the test set and an accuracy of 94.98% on 5-fold cross-validation. During this approach, we applied a count vectorizer and n-grams of (1,1). Stacking, is an ensemble learning technique in machine learning where multiple models are trained, and their predictions are combined to generate a final output [16]. Thus, we applied Stacking Multinomial Naïve Bayes and Multi-layer Perceptron classifier using chi-squared feature selection. It has high accuracy in both k-Fold Cross-Validation and in the test set. Another approach that led to high accuracy on k-fold cross-fold validation and test was Stacking Logistic Regression and Multinomial Naïve Bayes. However, none of these two algorithms could overtake the Multi-layer Perception accuracy. All these results are depicted in Table 1.

6 Future Work

We have provided some methods which could be useful for further investigations:

- **Deep learning models:** some of the deep learning models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) proved to supply a great performance in text analysis tasks [17].
- **Different feature extraction techniques:** word embeddings or other vectorization methods can be experimented with . in natural language processing (NLP), embedding is a numerical representation of words or text that captures their meaning in a lower-dimensional space. This representation is learned from large amounts of text data using neural network models, such as word2vec, GloVe, and FastText [18].

7 Statement of Contributions

Mahta Amini: data preparation, algorithm implementation, model evaluation and report preparation.
Sara Yabesi: data preparation, algorithm implementation, model evaluation and report preparation.
Baharan Nouriinanloo: data preparation, algorithm implementation, model evaluation and report preparation.

References

[1] M Ikonomakis, Sotiris Kotsiantis, and Vasilis Tampakas. Text classification using machine learning techniques. *WSEAS transactions on computers*, 4(8):966–974, 2005.

[2] Joël Plisson, Nada Lavrac, Dunja Mladenic, et al. A rule based approach to word lemmatization. In *Proceedings of IS*, volume 3, pages 83–86, 2004.

[3] Peter Willett. The porter stemming algorithm: then and now. *Program*, 40(3):219–223, 2006.

[4] Todd Michael Franke, Timothy Ho, and Christina A Christie. The chi-square test: Often used and more often misinterpreted. *American journal of evaluation*, 33(3):448–458, 2012.

[5] Huawen Liu, Jigui Sun, Lei Liu, and Huijie Zhang. Feature selection with dynamic mutual information. *Pattern Recognition*, 42(7):1330–1339, 2009.

[6] Shuo Xu. Bayesian naïve bayes classifiers to text classification. *Journal of Information Science*, 44(1):48–59, 2018.

[7] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021.

[8] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[9] Amir Ali Pour, David Hely, Vincent Beroulle, and Giorgio Di Natale. An efficient approach to model strong puf with multi-layer perceptron using transfer learning. In *International Symposium on Quality Electronic Design (ISQED 2022)*. IEEE, 2022.

[10] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1249, 2018.

[11] GSK Ranjan, Amar Kumar Verma, and Sudha Radhika. K-nearest neighbors and grid search cv based real time fault monitoring system for industries. In *2019 IEEE 5th international conference for convergence in technology (I2CT)*, pages 1–5. IEEE, 2019.

[12] Edward Loper and Steven Bird. Nltk: The natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[13] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, volume 161175. Las Vegas, NV, 1994.

[14] Sinan Ozdemir and Divya Susarla. *Feature Engineering Made Easy: Identify unique features from your dataset in order to build powerful machine learning systems*. Packt Publishing Ltd, 2018.

[15] Hae-Young Kim. Analysis of variance (anova) comparing means of more than two groups. *Restorative dentistry & endodontics*, 39(1):74–77, 2014.

[16] Fahima Hossain, Mohammed Nasir Uddin, and Rajib Kumar Halder. An ensemble method-based machine learning approach using text mining to identify semantic fake news. In *Proceedings of the International Conference on Big Data, IoT, and Machine Learning: BIM 2021*, pages 733–744. Springer, 2022.

[17] Wang Wang, Guangze Wen, and Zikun Zheng. Design of deep learning mixed language short text sentiment classification system based on cnn algorithm. In *2022 IEEE 2nd International Conference on Mobile Networks and Wireless Communications (ICMNWC)*, pages 1–5. IEEE, 2022.

[18] EDDY MUNTINA Dharma, F Lumban Gaol, HLHS Warnars, and BENFANO Soewito. The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification. *Journal of Theoretical and Applied Information Technology*, 100(2):31, 2022.

8 Appendix

BernoulliNaiveBayes_Custom

March 12, 2023

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from google.colab import drive
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import random
import time
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2,␣
 ↪f_classif,mutual_info_classif
from sklearn.preprocessing import Normalizer
from sklearn import model_selection
from sklearn import svm
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from enum import Enum

nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[2]:
```
True
```

[3]:
```
#import the data
drive.mount('/content/gdrive/', force_remount=True)

train_data_initial = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/train.
 ↪csv')
test_data = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/test.csv')

print('shape train:',train_data_initial.shape)
print('shape test:',test_data.shape)
```

```
Mounted at /content/gdrive/
shape train: (718, 2)
shape test: (279, 2)
```

[4]:
```
train_data_initial.columns.values.tolist()
```

[4]:
```
['body', 'subreddit']
```

[5]:
```
train_data_initial.describe()
```

[5]:

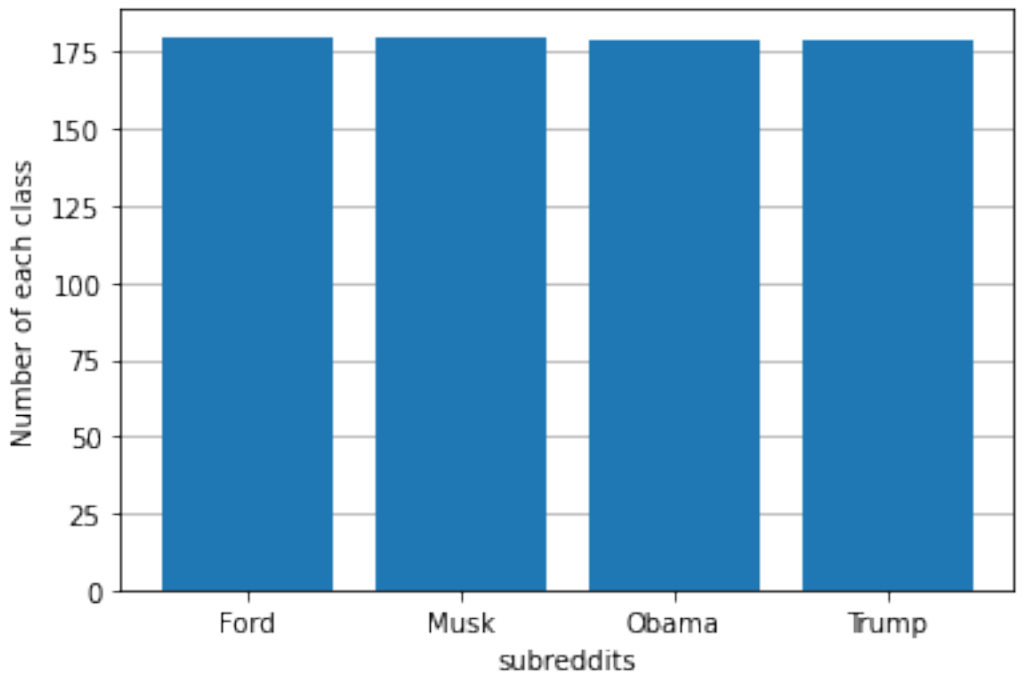
	body	subreddit
count	718	718
unique	636	4
top	Hi there /u/LakotaPride! Welcome to /r/Trump. …	Obama
freq	30	180

[5]:
```
#distribution of each subreddit
train_label_counts = train_data_initial["subreddit"].value_counts()
print(train_label_counts)
unique_labels = train_data_initial["subreddit"].unique()

fig, ax = plt.subplots()
ax.grid(zorder=1, axis="y")
ax.bar(unique_labels, train_label_counts, zorder=2)
ax.set_xticks([0,1,2,3])
ax.set_xticklabels(unique_labels)
ax.set_ylabel("Number of each class")
ax.set_xlabel("subreddits")
```

```
Obama    180
Trump    180
Ford     179
Musk     179
Name: subreddit, dtype: int64
```

[5]: `Text(0.5, 0, 'subreddits')`



[6]:
```
#check if tehre are any duplicates or isnull values,min and max
train_data_initial.duplicated().sum()
print("is duplicate:",train_data_initial.duplicated().sum())

print("is null:",train_data_initial.isnull().values.any())

print("maxmium values",train_data_initial.max(axis=0))
print("minimum values",train_data_initial.min(axis=0))
```

```
is duplicate: 82
is null: False
maxmium values body         you can collect free energy right now…with a…
subreddit                                                       Trump
dtype: object
minimum values body         \nIf the link is behind a paywall, or for an a…
subreddit                                                        Ford
dtype: object
```

[7]:
```
#describe the train data
train_data_initial['body'].apply(lambda x: len(x.split())).describe()
```

[7]:
```
count    718.000000
mean     102.512535
std      104.806705
min        1.000000
25%       56.250000
50%       60.000000
75%       90.500000
max      557.000000
Name: body, dtype: float64
```

[8]:
```
#describe the test data
test_data['body'].apply(lambda x: len(x.split())).describe()
```

[8]:
```
count    279.000000
mean     103.426523
std      108.965248
min        1.000000
25%       59.000000
50%       60.000000
75%       87.000000
max      496.000000
Name: body, dtype: float64
```

[6]:
```
def shuffle_data(df):
    random.seed(0)  # Use a fixed seed for the random number generator
```

```
    df = df.sample(frac=1, random_state=0).reset_index(drop=True)
    return df
```

[7]:
```
#shuffle the data and split the features from the label
train_data = shuffle_data(train_data_initial)

train_x = train_data["body"]
train_y = train_data["subreddit"]
test_x = test_data["body"]
```

[11]:
```
print('train_data shape=>',train_data.shape)
print('train_data x=>',train_x.shape)
print('train_data y=>',train_y.shape)
```

```
train_data shape=> (718, 2)
train_data x=> (718,)
train_data y=> (718,)
```

[12]:
```
print(train_y.head())
```

```
0    Obama
1    Trump
2     Musk
3     Ford
4    Obama
Name: subreddit, dtype: object
```

[8]:
```
#stop words
sklearn_stop_words = text.ENGLISH_STOP_WORDS
print(len(sklearn_stop_words))
stop_words_nltk = set(stopwords.words('english'))
print(len(stop_words_nltk))
```

```
318
179
```

[9]:
```
#prior class probability can either be learned or a uniform priority be used
prior_probabilities = Enum('prior_probabilities', ['learn', 'uniform'])
```

[10]:
```
#function for creating the test csv file to upload to kaggle
def create_test_csv(data, outfile_name):
  rawdata= {'subreddit':data}
  csv = pd.DataFrame(rawdata, columns = ['subreddit'])
  csv.to_csv(outfile_name,index=True, header=True)
  print ("File saved.")
```

[11]:
```
class CustomNaiveBayes:
    def __init__(self, alpha=0.01, prior=prior_probabilities.learn):
        self.alpha = alpha
        self.prior = prior

    #fit function
    def fit(self, X, y):
        X = X.toarray()
        class_counts = y.value_counts()

        num_labels = len(class_counts)

        #calculate prior probability
        if self.prior == prior_probabilities.learn:
          self.class_probabilities = class_counts / len(y)
        elif self.prior == prior_probabilities.uniform:
           self.class_probabilities = pd.Series(np.repeat(1/num_labels,␣
 ↪num_labels),
                                                index = class_counts.index)

        self.class_probabilities.sort_index(inplace=True)
        class_counts.sort_index(inplace=True)
        features_count = np.empty((num_labels, X.shape[1]))
        y_numpy = y.to_numpy()

        for i in range(num_labels):

            label = self.class_probabilities.index[i]
            X_this_label = X[np.nonzero(y_numpy == label), :]

            features_count[i,:] = np.sum(X_this_label, axis=1)

        # add Laplace smoothing
        smoothed_numerator = features_count + self.alpha
        smoothed_denominator = np.sum(smoothed_numerator,axis=1).reshape(-1,1)
        self.parameters = pd.DataFrame(smoothed_numerator /␣
 ↪smoothed_denominator, index=self.class_probabilities.index)

    #predict function
    def predict(self, X):

        X = X.toarray()
        delta = pd.DataFrame(columns=self.class_probabilities.index)
        for label in self.class_probabilities.index:

            class_probability = self.class_probabilities[label]
            theta_j_class = self.parameters.loc[label, :].to_numpy()
```

```
            prob_features_given_y = (theta_j_class ** X) * (1 - theta_j_class)␣
 ↪** (
                1 - X
            )
            prob_sample_given_y = np.prod(prob_features_given_y, axis=1)
            # Compute final probability
            term1 = np.log(class_probability)
            term2 = np.sum(X * np.log(theta_j_class), axis=1)
            term3 = np.sum((1 - X) * np.log(1 - theta_j_class), axis=1)
            delta_k = term1 + term2 + term3
            delta[label] = delta_k

        predicted_class = delta.idxmax(axis=1)
        return predicted_class.to_list()

    #score function
    def score(self, X, y):
        y_pred = self.predict(X)
        accuracy = np.count_nonzero(y == y_pred) / len(y_pred)
        return accuracy


    #get parameters function
    def get_params(self, deep=True):
        params = {"alpha": self.alpha,
                  "prior": self.prior}
        return params

    #set parameters function
    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self
```

[12]:
```
#start testing different modes
#remove punctuation
import re
import string

data_x_punc = train_x.copy()

for i in range(data_x_punc.shape[0]):
  data_x_punc[i]= data_x_punc[i].translate(str.maketrans('', '', string.
 ↪punctuation))
```

```
test_x_punc = test_x.copy()

for i in range(test_x_punc.shape[0]):
  test_x_punc[i]= test_x_punc[i].translate(str.maketrans('', '', string.
 ↪punctuation))
```

[20]:
```
print(data_x_punc.shape)
print(train_x.shape)
print(test_x_punc.shape)
```

```
(718,)
(718,)
(279,)
```

[13]:
```
#create a dictionary of stop words
stop_words_nltk = set(stopwords.words('english'))
print(len(stop_words_nltk))
stop_words_sklearn = text.ENGLISH_STOP_WORDS
print(len(stop_words_sklearn))

stop_words_custom = [
    # All pronouns and associated words
    "i","i'll","i'd","i'm","i've","ive","me","myself","you",
    "you'll",
    "you'd",
    "you're",
    "you've",
    "yourself",
    "he",
    "he'll",
    "he'd",
    "he's",
    "him",
    "she",
    "she'll",
    "she'd",
    "she's",
    "her",
    "it",
    "it'll",
    "it'd",
    "it's",
    "itself",
    "oneself",
    "we",
    "we'll",
    "we'd",
```

```
    "we're",
    "we've",
    "us",
    "ourselves",
    "they",
    "they'll",
    "they'd",
    "they're",
    "they've",
    "them",
    "themselves",
    "everyone",
    "everyone's",
    "everybody",
    "everybody's",
    "someone",
    "someone's",
    "somebody",
    "somebody's",
    "nobody",
    "nobody's",
    "anyone",
    "anyone's",
    "everything",
    "everything's",
    "something",
    "something's",
    "nothing",
    "nothing's",
    "anything",
    "anything's",
    # All determiners and associated words
    "a",
    "an",
    "the",
    "this",
    "that",
    "that's",
    "these",
    "those",
    "my",
    #"mine",   #Omitted since mine can refer to something else
    "your",
    "yours",
    "his",
    "hers",
    "its",
```

```
    "our",
    "ours",
    "own",
    "their",
    "theirs",
    "few",
    "much",
    "many",
    "lot",
    "lots",
    "some",
    "any",
    "enough",
    "all",
    "both",
    "half",
    "either",
    "neither",
    "each",
    "every",
    "certain",
    "other",
    "another",
    "such",
    "several",
    "multiple",
    # "what",    #Dealt with later on
    "rather",
    "quite",
    # All prepositions
    "aboard",
    "about",
    "above",
    "across",
    "after",
    "against",
    "along",
    "amid",
    "amidst",
    "among",
    "amongst",
    "anti",
    "around",
    "as",
    "at",
    "away",
    "before",
```

```
    "behind",
    "below",
    "beneath",
    "beside",
    "besides",
    "between",
    "beyond",
    "but",
    "by",
    "concerning",
    "considering",
    "despite",
    "down",
    "during",
    "except",
    "excepting",
    "excluding",
    "far",
    "following",
    "for",
    "from",
    "here",
    "here's",
    "in",
    "inside",
    "into",
    "left",
    "like",
    "minus",
    "near",
    "of",
    "off",
    "on",
    "onto",
    "opposite",
    "out",
    "outside",
    "over",
    "past",
    "per",
    "plus",
    "regarding",
    "right",
    #"round",    #Omitted
    #"save",     #Omitted
    "since",
    "than",
```

```
    "there",
    "there's",
    "through",
    "to",
    "toward",
    "towards",
    "under",
    "underneath",
    "unlike",
    "until",
    "up",
    "upon",
    "versus",
    "via",
    "with",
    "within",
    "without",
    # Irrelevant verbs
    "may",
    "might",
    "will",
    "won't",
    "would",
    "wouldn't",
    "can",
    "can't",
    "cannot",
    "could",
    "couldn't",
    "should",
    "shouldn't",
    "must",
    "must've",
    "be",
    "being",
    "been",
    "am",
    "are",
    "aren't",
    "ain't",
    "is",
    "isn't",
    "was",
    "wasn't",
    "were",
    "weren't",
    "do",
```

```
    "doing",
    "don't",
    "does",
    "doesn't",
    "did",
    "didn't",
    "done",
    "have",
    "haven't",
    "having",
    "has",
    "hasn't",
    "had",
    "hadn't",
    "get",
    "getting",
    "gets",
    "got",
    "gotten",
    "go",
    "going",
    "gonna",
    "goes",
    "went",
    "gone",
    "make",
    "making",
    "makes",
    "made",
    "take",
    "taking",
    "takes",
    "took",
    "taken",
    "need",
    "needing",
    "needs",
    "needed",
    "use",
    "using",
    "uses",
    "used",
    "want",
    "wanna",
    "wanting",
    "wants",
    "let",
```

```
    "lets",
    "letting",
    "let's",
    "suppose",
    "supposing",
    "supposes",
    "supposed",
    "seem",
    "seeming",
    "seems",
    "seemed",
    "say",
    "saying",
    "says",
    "said",
    "know",
    "knowing",
    "knows",
    "knew",
    "known",
    "look",
    "looking",
    "looked",
    "think",
    "thinking",
    "thinks",
    "thought",
    "feel",
    "feels",
    "felt",
    "based",
    "put",
    "puts",
    #"wanted"   #Omitted since the advective is relevant
    # Question words and associated words
    "who",
    "who's",
    "who've",
    "who'd",
    "whoever",
    "whoever's",
    "whom",
    "whomever",
    "whomever's",
    "whose",
    "whosever",
    "whosever's",
```

```
    "when",
    "whenever",
    "which",
    "whichever",
    "where",
    "where's",
    "where'd",
    "wherever",
    "why",
    "why's",
    "why'd",
    "whyever",
    "what",
    "what's",
    "whatever",
    "whence",
    "how",
    "how's",
    "how'd",
    "however",
    "whether",
    "whatsoever",
    # Connector words and irrelevant adverbs
    "and",
    "or",
    "not",
    "because",
    "also",
    "always",
    "never",
    "only",
    "really",
    "very",
    "greatly",
    "extremely",
    "somewhat",
    "no",
    "nope",
    "nah",
    "yes",
    "yep",
    "yeh",
    "yeah",
    "maybe",
    "perhaps",
    "more",
    "most",
```

```
    "less",
    "least",
    "good",
    "great",
    "well",
    "better",
    "best",
    "bad",
    "worse",
    "worst",
    "too",
    "thru",
    "though",
    "although",
    "yet",
    "already",
    "then",
    "even",
    "now",
    "sometimes",
    "still",
    "together",
    "altogether",
    "entirely",
    "fully",
    "entire",
    "whole",
    "completely",
    "utterly",
    "seemingly",
    "apparently",
    "clearly",
    "obviously",
    "actually",
    "actual",
    "usually",
    "usual",
    "literally",
    "honestly",
    "absolutely",
    "definitely",
    "generally",
    "totally",
    "finally",
    "basically",
    "essentially",
    "fundamentally",
```

```
    "automatically",
    "immediately",
    "necessarily",
    "primarily",
    "normally",
    "perfectly",
    "constantly",
    "particularly",
    "eventually",
    "hopefully",
    "mainly",
    "typically",
    "specifically",
    "differently",
    "appropriately",
    "plenty",
    "certainly",
    "unfortunately",
    "ultimately",
    "unlikely",
    "likely",
    "potentially",
    "fortunately",
    "personally",
    "directly",
    "indirectly",
    "nearly",
    "closely",
    "slightly",
    "probably",
    "possibly",
    "especially",
    "frequently",
    "often",
    "oftentimes",
    "seldom",
    "rarely",
    "sure",
    "while",
    "whilst",
    "able",
    "unable",
    "else",
    "ever",
    "once",
    "twice",
    "thrice",
```

```
    "almost",
    "again",
    "instead",
    "next",
    "previous",
    "unless",
    "somehow",
    "anyhow",
    "anywhere",
    "somewhere",
    "everywhere",
    "nowhere",
    "further",
    "anymore",
    "later",
    "ago",
    "ahead",
    "just",
    "same",
    "different",
    "big",
    "small",
    "little",
    "tiny",
    "large",
    "huge",
    "pretty",
    "mostly",
    "anyway",
    "anyways",
    "otherwise",
    "regardless",
    "throughout",
    "additionally",
    "moreover",
    "furthermore",
    "meanwhile",
    "afterwards",
    # Irrelevant nouns
    "thing",
    "thing's",
    "things",
    "stuff",
    "other's",
    "others",
    "another's",
    "total",
```

```
    "",
    "false",
    "none",
    "way",
    "kind",
    # Lettered numbers and order
    "zero",
    "zeros",
    "zeroes",
    "one",
    "ones",
    "two",
    "three",
    "four",
    "five",
    "six",
    "seven",
    "eight",
    "nine",
    "ten",
    "twenty",
    "thirty",
    "forty",
    "fifty",
    "sixty",
    "seventy",
    "eighty",
    "ninety",
    "hundred",
    "hundreds",
    "thousand",
    "thousands",
    "million",
    "millions",
    "first",
    "last",
    "second",
    "third",
    "fourth",
    "fifth",
    "sixth",
    "seventh",
    "eigth",
    "ninth",
    "tenth",
    "firstly",
    "secondly",
```

```
    "thirdly",
    "lastly",
    # Greetings and slang
    "hello",
    "hi",
    "hey",
    "sup",
    "yo",
    "greetings",
    "please",
    "okay",
    "ok",
    "y'all",
    "lol",
    "rofl",
    "thank",
    "thanks",
    "alright",
    "kinda",
    "dont",
    "sorry",
    "idk",
    "tldr",
    "tl",
    "dr",  #This means that dr (doctor) is a bad feature because of tl;dr
    "tbh",
    "dude",
    "tho",
    "aka",
    "plz",
    "pls",
    "bit",
    "don",
    # Miscellaneous
    "www",
    "https",
    "http",
    "com",
    "etc",
    "html",
    "reddit",
    "subreddit",
    "subreddits",
    "comments",
    "reply",
    "replies",
    "thread",
```

```
    "threads",
    "post",
    "posts",
    "website",
    "websites",
    "web site",
    "web sites"]
print('length custom:',len(stop_words_custom))
```

```
179
318
length custom: 590
```

[14]:
```
#remove punctuation from custom stop words
stop_words_custom_punc = stop_words_custom
print(len(stop_words_custom_punc))
for i in range(len(stop_words_custom_punc)):
 stop_words_custom_punc[i]= stop_words_custom_punc[i].translate(str.
 ↪maketrans('', '', string.punctuation))
```

```
590
```

[15]:
```
#lemmatization
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

class LemmaTokenizer_Pos:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in␣
 ↪word_tokenize(doc) if t.isalpha()]

class LemmaTokenizer:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) if t.
 ↪isalpha()]

class LemmaTokenizer_word:
```

```
    def __init__(self):
      self.wnl = WordNetLemmatizer()
    def __call__(self, doc):
      return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) ]


class StemTokenizer:
    def __init__(self):
      self.wnl =PorterStemmer()
    def __call__(self, doc):
      return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
```

[25]:
```
#start testing different modes
#select alpha

t_start = time.time()

pipe_params = {
    "selecter__k":[5000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5],
    "vect__stop_words": [stop_words_custom_punc],
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
normalizer = Normalizer()
model = CustomNaiveBayes()

pipe = Pipeline(
    [("vect", vectorizer), ("norm", normalizer), ("selecter", selecter),␣
 ↪("classify", CustomNaiveBayes())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()


print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
The best accuracy is 0.9303321678321679.
The winning parameters are {'classify__alpha': 0.02, 'selecter__k': 5000,
```

```
'vect__stop_words': ['i', 'ill', 'id', 'im', 'ive', 'ive', 'me', 'myself',
'you', 'youll', 'youd', 'youre', 'youve', 'yourself', 'he', 'hell', 'hed',
'hes', 'him', 'she', 'shell', 'shed', 'shes', 'her', 'it', 'itll', 'itd', 'its',
'itself', 'oneself', 'we', 'well', 'wed', 'were', 'weve', 'us', 'ourselves',
'they', 'theyll', 'theyd', 'theyre', 'theyve', 'them', 'themselves', 'everyone',
'everyones', 'everybody', 'everybodys', 'someone', 'someones', 'somebody',
'somebodys', 'nobody', 'nobodys', 'anyone', 'anyones', 'everything',
'everythings', 'something', 'somethings', 'nothing', 'nothings', 'anything',
'anythings', 'a', 'an', 'the', 'this', 'that', 'thats', 'these', 'those', 'my',
'your', 'yours', 'his', 'hers', 'its', 'our', 'ours', 'own', 'their', 'theirs',
'few', 'much', 'many', 'lot', 'lots', 'some', 'any', 'enough', 'all', 'both',
'half', 'either', 'neither', 'each', 'every', 'certain', 'other', 'another',
'such', 'several', 'multiple', 'rather', 'quite', 'aboard', 'about', 'above',
'across', 'after', 'against', 'along', 'amid', 'amidst', 'among', 'amongst',
'anti', 'around', 'as', 'at', 'away', 'before', 'behind', 'below', 'beneath',
'beside', 'besides', 'between', 'beyond', 'but', 'by', 'concerning',
'considering', 'despite', 'down', 'during', 'except', 'excepting', 'excluding',
'far', 'following', 'for', 'from', 'here', 'heres', 'in', 'inside', 'into',
'left', 'like', 'minus', 'near', 'of', 'off', 'on', 'onto', 'opposite', 'out',
'outside', 'over', 'past', 'per', 'plus', 'regarding', 'right', 'since', 'than',
'there', 'theres', 'through', 'to', 'toward', 'towards', 'under', 'underneath',
'unlike', 'until', 'up', 'upon', 'versus', 'via', 'with', 'within', 'without',
'may', 'might', 'will', 'wont', 'would', 'wouldnt', 'can', 'cant', 'cannot',
'could', 'couldnt', 'should', 'shouldnt', 'must', 'mustve', 'be', 'being',
'been', 'am', 'are', 'arent', 'aint', 'is', 'isnt', 'was', 'wasnt', 'were',
'werent', 'do', 'doing', 'dont', 'does', 'doesnt', 'did', 'didnt', 'done',
'have', 'havent', 'having', 'has', 'hasnt', 'had', 'hadnt', 'get', 'getting',
'gets', 'got', 'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make',
'making', 'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need',
'needing', 'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna',
'wanting', 'wants', 'let', 'lets', 'letting', 'lets', 'suppose', 'supposing',
'supposes', 'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying',
'says', 'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', 'whos', 'whove', 'whod', 'whoever', 'whoevers',
'whom', 'whomever', 'whomevers', 'whose', 'whosever', 'whosevers', 'when',
'whenever', 'which', 'whichever', 'where', 'wheres', 'whered', 'wherever',
'why', 'whys', 'whyd', 'whyever', 'what', 'whats', 'whatever', 'whence', 'how',
'hows', 'howd', 'however', 'whether', 'whatsoever', 'and', 'or', 'not',
'because', 'also', 'always', 'never', 'only', 'really', 'very', 'greatly',
'extremely', 'somewhat', 'no', 'nope', 'nah', 'yes', 'yep', 'yeh', 'yeah',
'maybe', 'perhaps', 'more', 'most', 'less', 'least', 'good', 'great', 'well',
'better', 'best', 'bad', 'worse', 'worst', 'too', 'thru', 'though', 'although',
'yet', 'already', 'then', 'even', 'now', 'sometimes', 'still', 'together',
'altogether', 'entirely', 'fully', 'entire', 'whole', 'completely', 'utterly',
'seemingly', 'apparently', 'clearly', 'obviously', 'actually', 'actual',
'usually', 'usual', 'literally', 'honestly', 'absolutely', 'definitely',
'generally', 'totally', 'finally', 'basically', 'essentially', 'fundamentally',
```

```
'automatically', 'immediately', 'necessarily', 'primarily', 'normally',
'perfectly', 'constantly', 'particularly', 'eventually', 'hopefully', 'mainly',
'typically', 'specifically', 'differently', 'appropriately', 'plenty',
'certainly', 'unfortunately', 'ultimately', 'unlikely', 'likely', 'potentially',
'fortunately', 'personally', 'directly', 'indirectly', 'nearly', 'closely',
'slightly', 'probably', 'possibly', 'especially', 'frequently', 'often',
'oftentimes', 'seldom', 'rarely', 'sure', 'while', 'whilst', 'able', 'unable',
'else', 'ever', 'once', 'twice', 'thrice', 'almost', 'again', 'instead', 'next',
'previous', 'unless', 'somehow', 'anyhow', 'anywhere', 'somewhere',
'everywhere', 'nowhere', 'further', 'anymore', 'later', 'ago', 'ahead', 'just',
'same', 'different', 'big', 'small', 'little', 'tiny', 'large', 'huge',
'pretty', 'mostly', 'anyway', 'anyways', 'otherwise', 'regardless',
'throughout', 'additionally', 'moreover', 'furthermore', 'meanwhile',
'afterwards', 'thing', 'things', 'things', 'stuff', 'others', 'others',
'anothers', 'total', '', 'false', 'none', 'way', 'kind', 'zero', 'zeros',
'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six', 'seven',
'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', 'yall', 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etc', 'html', 'reddit',
'subreddit', 'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads',
'post', 'posts', 'website', 'websites', 'web site', 'web sites']}
Run time:  4.818 seconds

/usr/local/lib/python3.9/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['site', 'sites', 'web'] not in
stop_words.
  warnings.warn(
```

[26]:
```
#start testing different modes
#remove normalize
pipe_params = {
    "selecter__k":[5000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
normalizer = Normalizer()
model = CustomNaiveBayes()

pipe = Pipeline(
```

```
    [("vect", vectorizer), ("norm", normalizer), ("selecter", selecter),␣
 ↪("classify", CustomNaiveBayes())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
The best accuracy is 0.8996697746697746.
The winning parameters are {'classify__alpha': 0.001, 'selecter__k': 5000}
```

[27]:
```
#start testing different modes
#stop words - scikitlearn
t_start = time.time()

pipe_params = {
    "selecter__k":[5000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = CountVectorizer(stop_words=list(stop_words_sklearn))
selecter = SelectKBest(chi2)
model = CustomNaiveBayes()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter), ("classify",␣
 ↪CustomNaiveBayes())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
The best accuracy is 0.9178321678321678.
The winning parameters are {'classify__alpha': 0.1, 'selecter__k': 5000}
Run time:  6.200 seconds
```

[29]:
```
#stop words - nltk
t_start = time.time()

pipe_params = {
    "selecter__k":[5000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = CountVectorizer(stop_words=list(stop_words_nltk))
selecter = SelectKBest(chi2)
model = CustomNaiveBayes()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter), ("classify",␣
 ↪CustomNaiveBayes())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
The best accuracy is 0.910878010878011.
The winning parameters are {'classify__alpha': 0.01, 'selecter__k': 5000}
Run time:  4.087 seconds
```

[27]:
```
#stop words - custom
pipe_params = {
    "selecter__k":[5000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

t_start = time.time()

vectorizer = CountVectorizer(stop_words=list(stop_words_custom_punc))
selecter = SelectKBest(chi2)
model = CustomNaiveBayes()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter), ("classify",␣
 ↪CustomNaiveBayes())]
```

```
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
The best accuracy is 0.9359168609168609.
The winning parameters are {'classify__alpha': 0.5, 'selecter__k': 5000}
Run time:  3.791 seconds
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['site', 'sites', 'web'] not in
stop_words.
  warnings.warn(
```

[31]:
```
stop_words_library = stop_words_sklearn.union(stop_words_nltk)
final_stop_words =  set(stop_words_custom) | stop_words_library
```

[28]:
```
#create new stop word dictionary
stop_words_library = stop_words_sklearn.union(stop_words_nltk)
final_stop_words =  set(stop_words_custom_punc) | stop_words_library
t_start = time.time()


pipe_params = {
    "selecter__k":[5000,3000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = CountVectorizer(stop_words=list(final_stop_words))
selecter = SelectKBest(chi2)
model = CustomNaiveBayes()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter), ("classify",␣
 ↪CustomNaiveBayes())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)
```

```
grid.fit(train_x, train_y)

t_end = time.time()

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
The best accuracy is 0.9429001554001555.
The winning parameters are {'classify__alpha': 0.5, 'selecter__k': 5000}
Run time:  7.653 seconds
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['site', 'sites', 'web'] not in
stop_words.
  warnings.warn(
```

[29]:
```
#generate test.csv
y_pred = grid.predict(test_x)

create_test_csv(y_pred,"customNaiveBayes_12032023_02.csv")
```

```
File saved.
```

[34]:
```
#final stop words - test normalize
t_start = time.time()

pipe_params = {
    "selecter__k":[5000,3000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = CountVectorizer(stop_words=list(final_stop_words))
selecter = SelectKBest(chi2)
model = CustomNaiveBayes()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter) ,("classify", model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()
```

```
print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
The best accuracy is 0.9429001554001555.
The winning parameters are {'classify__alpha': 0.5, 'selecter__k': 5000}
Run time:  9.414 seconds
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['site', 'sites', 'web'] not in
stop_words.
  warnings.warn(
```

[36]:
```
#test lemmatizer- LemmaTokenizer_Pos

t_start = time.time()

pipe_params = {
    "selecter__k":[5000,3000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer =␣
 ↪CountVectorizer(stop_words=list(final_stop_words),tokenizer=LemmaTokenizer())
selecter = SelectKBest(chi2)
model = CustomNaiveBayes()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter) ,("classify", model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
/usr/local/lib/python3.9/dist-
```

```
packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
25 fits failed out of a total of 50.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 401,
in fit
    Xt = self._fit(X, y, **fit_params_steps)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 359,
in _fit
    X, fitted_transformer = fit_transform_one_cached(
  File "/usr/local/lib/python3.9/dist-packages/joblib/memory.py", line 349, in
__call__
    return self.func(*args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 893,
in _fit_transform_one
    res = transformer.fit_transform(X, y, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/utils/_set_output.py",
line 142, in wrapped
    data_to_wrap = f(self, X, *args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/base.py", line 862, in
fit_transform
    return self.fit(X, y, **fit_params).transform(X)
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/feature_selection/_univariate_selection.py", line 471, in fit
    self._check_params(X, y)
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/feature_selection/_univariate_selection.py", line 672, in
_check_params
    raise ValueError(
ValueError: k should be <= n_features = 4794; got 5000. Use k='all' to return
all features.

--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
```

```
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 401,
in fit
    Xt = self._fit(X, y, **fit_params_steps)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 359,
in _fit
    X, fitted_transformer = fit_transform_one_cached(
  File "/usr/local/lib/python3.9/dist-packages/joblib/memory.py", line 349, in
__call__
    return self.func(*args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 893,
in _fit_transform_one
    res = transformer.fit_transform(X, y, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/utils/_set_output.py",
line 142, in wrapped
    data_to_wrap = f(self, X, *args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/base.py", line 862, in
fit_transform
    return self.fit(X, y, **fit_params).transform(X)
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/feature_selection/_univariate_selection.py", line 471, in fit
    self._check_params(X, y)
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/feature_selection/_univariate_selection.py", line 672, in
_check_params
    raise ValueError(
ValueError: k should be <= n_features = 4751; got 5000. Use k='all' to return
all features.

--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 401,
in fit
    Xt = self._fit(X, y, **fit_params_steps)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 359,
in _fit
    X, fitted_transformer = fit_transform_one_cached(
  File "/usr/local/lib/python3.9/dist-packages/joblib/memory.py", line 349, in
__call__
    return self.func(*args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 893,
in _fit_transform_one
    res = transformer.fit_transform(X, y, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/utils/_set_output.py",
line 142, in wrapped
```

```
    data_to_wrap = f(self, X, *args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/base.py", line 862, in
fit_transform
    return self.fit(X, y, **fit_params).transform(X)
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/feature_selection/_univariate_selection.py", line 471, in fit
    self._check_params(X, y)
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/feature_selection/_univariate_selection.py", line 672, in
_check_params
    raise ValueError(
ValueError: k should be <= n_features = 4832; got 5000. Use k='all' to return
all features.

--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 401,
in fit
    Xt = self._fit(X, y, **fit_params_steps)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 359,
in _fit
    X, fitted_transformer = fit_transform_one_cached(
  File "/usr/local/lib/python3.9/dist-packages/joblib/memory.py", line 349, in
__call__
    return self.func(*args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 893,
in _fit_transform_one
    res = transformer.fit_transform(X, y, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/utils/_set_output.py",
line 142, in wrapped
    data_to_wrap = f(self, X, *args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/base.py", line 862, in
fit_transform
    return self.fit(X, y, **fit_params).transform(X)
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/feature_selection/_univariate_selection.py", line 471, in fit
    self._check_params(X, y)
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/feature_selection/_univariate_selection.py", line 672, in
_check_params
    raise ValueError(
ValueError: k should be <= n_features = 4884; got 5000. Use k='all' to return
all features.
```

```
--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 401,
in fit
    Xt = self._fit(X, y, **fit_params_steps)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 359,
in _fit
    X, fitted_transformer = fit_transform_one_cached(
  File "/usr/local/lib/python3.9/dist-packages/joblib/memory.py", line 349, in
__call__
    return self.func(*args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/pipeline.py", line 893,
in _fit_transform_one
    res = transformer.fit_transform(X, y, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/utils/_set_output.py",
line 142, in wrapped
    data_to_wrap = f(self, X, *args, **kwargs)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/base.py", line 862, in
fit_transform
    return self.fit(X, y, **fit_params).transform(X)
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/feature_selection/_univariate_selection.py", line 471, in fit
    self._check_params(X, y)
  File "/usr/local/lib/python3.9/dist-
packages/sklearn/feature_selection/_univariate_selection.py", line 672, in
_check_params
    raise ValueError(
ValueError: k should be <= n_features = 4798; got 5000. Use k='all' to return
all features.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_search.py:952:
UserWarning: One or more of the test scores are non-finite: [       nan
0.90113636        nan 0.91085859        nan 0.91920163
        nan 0.91363636        nan 0.91640443]
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/feature_extraction/text.py:528:
UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is
not None'
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['base', 'bite', 'comment',
'concern', 'consider', 'exclude', 'follow', 'gon', 'greet', 'leave', 'na',
```

```
'regard', 'sha', 'sit', 'site', 'wan', 'web', 'win', 'wo'] not in stop_words.
  warnings.warn(

The best accuracy is 0.9192016317016318.
The winning parameters are {'classify__alpha': 0.1, 'selecter__k': 3000}
Run time:  43.618 seconds
```

[37]:
```
#test lemmatizer- LemmaTokenizer_word

t_start = time.time()

pipe_params = {
    "selecter__k":[5000,3000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer =␣
 ↪CountVectorizer(stop_words=list(final_stop_words),tokenizer=LemmaTokenizer_word())
selecter = SelectKBest(chi2)
model = CustomNaiveBayes()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter) ,("classify", model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

/usr/local/lib/python3.9/dist-packages/sklearn/feature_extraction/text.py:528:
UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is
not None'
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'ll", "'re", "'s", "'ve",
'base', 'bite', 'comment', 'concern', 'consider', 'exclude', 'follow', 'gon',
'greet', 'leave', "n't", 'na', 'regard', 'sha', 'sit', 'site', 'wan', 'web',
'win', 'wo'] not in stop_words.
  warnings.warn(
```

```
The best accuracy is 0.8983100233100233.
The winning parameters are {'classify__alpha': 0.01, 'selecter__k': 5000}
Run time:  50.826 seconds
```

[38]:
```python
#test stemmizer- StemTokenizer

t_start = time.time()

pipe_params = {

}
vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = CustomNaiveBayes()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter) ,("classify", model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=30)

grid.fit(train_x, train_y)

t_end = time.time()

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
The best accuracy is 0.72004662004662.
The winning parameters are {}
Run time:  6.797 seconds
```

[ ]:
```python
#test stemmizer and lemmatizer

t_start = time.time()

pipe_params = {
    "selecter__k":[5000,3000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5],
    "countVectorizer__tokenizer": [StemTokenizer(), LemmaTokenizer_word()]
}

vectorizer = CountVectorizer(stop_words=list(final_stop_words))
selecter = SelectKBest(chi2)
model = CustomNaiveBayes()
```

```
pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter) ,("classify", model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=1000)

grid.fit(train_x,train_y)

t_end = time.time()

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

[20]:
```
#test tfidf-without lemmarizer
import time

t_start = time.time()

pipe_params = {
    "selecter__k":[5000,3000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = TfidfVectorizer(stop_words=list(final_stop_words))
selecter = SelectKBest(chi2)
model = CustomNaiveBayes()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter) ,("classify", model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
The best accuracy is 0.9177641802641803.
The winning parameters are {'classify__alpha': 0.01, 'selecter__k': 5000}
Run time:  12.622 seconds
```

[21]:
```
#test tfidf-without lemmarizer-with normalize
import time

t_start = time.time()

pipe_params = {
    "selecter__k":[5000,3000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = TfidfVectorizer(stop_words=list(final_stop_words))
selecter = SelectKBest(chi2)
normalizer = Normalizer()
model = CustomNaiveBayes()


pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter), ("norm", normalizer)␣
 ↪,("classify", model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

print(f"The best accuracy is {grid.best_score_}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {t_end-t_start: .3f} seconds")
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
The best accuracy is 0.9177738927738928.
The winning parameters are {'classify__alpha': 0.1, 'selecter__k': 5000}
Run time:  16.289 seconds
```

[22]:
```
def print_best_params(grid):
  bestParameters = grid.best_estimator_.get_params()
  # print(bestParameters)
  for paramName in sorted(bestParameters.keys()):
    print("\t%s: %r" % (paramName, bestParameters[paramName]))
```

MLPClassifier

March 12, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from google.colab import drive
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import random
import time
import re
import string
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2,␣
 ↪f_classif,mutual_info_classif,f_regression
from sklearn.preprocessing import Normalizer
from sklearn import model_selection
from sklearn import svm
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[1]: True

[2]:
```
#import the data
drive.mount('/content/gdrive/', force_remount=True)

train_data_initial = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/train.
 ↪csv')
test_data = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/test.csv')

print('shape train:',train_data_initial.shape)
print('shape test:',test_data.shape)
```

```
Mounted at /content/gdrive/
shape train: (718, 2)
shape test: (279, 2)
```

[3]:
```
def shuffle_data(df):
    random.seed(0)  # Use a fixed seed for the random number generator
    df = df.sample(frac=1, random_state=0).reset_index(drop=True)
    return df
```

[4]:
```
#function for creating the test csv file to upload to kaggle
def create_test_csv(data, outfile_name):
  rawdata= {'subreddit':data}
  csv = pd.DataFrame(rawdata, columns = ['subreddit'])
  csv.to_csv(outfile_name,index=True, header=True)
```

```
  print ("File saved.")
```

[5]:
```
#shuffle the data and split the features from the label
train_data = shuffle_data(train_data_initial)

train_x = train_data["body"]
train_y = train_data["subreddit"]
test_x = test_data["body"]
```

[6]:
```
#remove punctuation
def remove_punctuation(text):
   translator = str.maketrans('', '', string.punctuation)
   text = text.translate(translator)
   return text
```

[7]:
```
#remove numeric values,lowercase words
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    return text
```

[9]:
```
def print_best_params(grid):
  bestParameters = grid.best_estimator_.get_params()
  for paramName in sorted(bestParameters.keys()):
    print("\t%s: %r" % (paramName, bestParameters[paramName]))
```

[10]:
```
#create a dictionary of stop words
stop_words_nltk = set(stopwords.words('english'))
stop_words_sklearn = text.ENGLISH_STOP_WORDS
stop_words_library = stop_words_sklearn.union(stop_words_nltk)
```

[11]:
```
#stemmer lemmatizer
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

class LemmaTokenizer_Pos:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in␣
 ↪word_tokenize(doc) if t.isalpha()]
```

```
class LemmaTokenizer:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) if t.
 ↪isalpha()]

class LemmaTokenizer_word:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) ]


class StemTokenizer:
     def __init__(self):
       self.wnl =PorterStemmer()
     def __call__(self, doc):
       return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
```

[12]:
```
stop_words_custom = [
    # All pronouns and associated words
    "i","i'll","i'd","i'm","i've","ive","me","myself","you",
    "you'll",
    "you'd",
    "you're",
    "you've",
    "yourself",
    "he",
    "he'll",
    "he'd",
    "he's",
    "him",
    "she",
    "she'll",
    "she'd",
    "she's",
    "her",
    "it",
    "it'll",
    "it'd",
    "it's",
    "itself",
    "oneself",
    "we",
    "we'll",
```

```
    "we'd",
    "we're",
    "we've",
    "us",
    "ourselves",
    "they",
    "they'll",
    "they'd",
    "they're",
    "they've",
    "them",
    "themselves",
    "everyone",
    "everyone's",
    "everybody",
    "everybody's",
    "someone",
    "someone's",
    "somebody",
    "somebody's",
    "nobody",
    "nobody's",
    "anyone",
    "anyone's",
    "everything",
    "everything's",
    "something",
    "something's",
    "nothing",
    "nothing's",
    "anything",
    "anything's",
    # All determiners and associated words
    "a",
    "an",
    "the",
    "this",
    "that",
    "that's",
    "these",
    "those",
    "my",
    #"mine",   #Omitted since mine can refer to something else
    "your",
    "yours",
    "his",
    "hers",
```

```
    "its",
    "our",
    "ours",
    "own",
    "their",
    "theirs",
    "few",
    "much",
    "many",
    "lot",
    "lots",
    "some",
    "any",
    "enough",
    "all",
    "both",
    "half",
    "either",
    "neither",
    "each",
    "every",
    "certain",
    "other",
    "another",
    "such",
    "several",
    "multiple",
    # "what",    #Dealt with later on
    "rather",
    "quite",
    # All prepositions
    "aboard",
    "about",
    "above",
    "across",
    "after",
    "against",
    "along",
    "amid",
    "amidst",
    "among",
    "amongst",
    "anti",
    "around",
    "as",
    "at",
    "away",
```

```
    "before",
    "behind",
    "below",
    "beneath",
    "beside",
    "besides",
    "between",
    "beyond",
    "but",
    "by",
    "concerning",
    "considering",
    "despite",
    "down",
    "during",
    "except",
    "excepting",
    "excluding",
    "far",
    "following",
    "for",
    "from",
    "here",
    "here's",
    "in",
    "inside",
    "into",
    "left",
    "like",
    "minus",
    "near",
    "of",
    "off",
    "on",
    "onto",
    "opposite",
    "out",
    "outside",
    "over",
    "past",
    "per",
    "plus",
    "regarding",
    "right",
    #"round",   #Omitted
    #"save",    #Omitted
    "since",
```

```
    "than",
    "there",
    "there's",
    "through",
    "to",
    "toward",
    "towards",
    "under",
    "underneath",
    "unlike",
    "until",
    "up",
    "upon",
    "versus",
    "via",
    "with",
    "within",
    "without",
    # Irrelevant verbs
    "may",
    "might",
    "will",
    "won't",
    "would",
    "wouldn't",
    "can",
    "can't",
    "cannot",
    "could",
    "couldn't",
    "should",
    "shouldn't",
    "must",
    "must've",
    "be",
    "being",
    "been",
    "am",
    "are",
    "aren't",
    "ain't",
    "is",
    "isn't",
    "was",
    "wasn't",
    "were",
    "weren't",
```

```
    "do",
    "doing",
    "don't",
    "does",
    "doesn't",
    "did",
    "didn't",
    "done",
    "have",
    "haven't",
    "having",
    "has",
    "hasn't",
    "had",
    "hadn't",
    "get",
    "getting",
    "gets",
    "got",
    "gotten",
    "go",
    "going",
    "gonna",
    "goes",
    "went",
    "gone",
    "make",
    "making",
    "makes",
    "made",
    "take",
    "taking",
    "takes",
    "took",
    "taken",
    "need",
    "needing",
    "needs",
    "needed",
    "use",
    "using",
    "uses",
    "used",
    "want",
    "wanna",
    "wanting",
    "wants",
```

```
    "let",
    "lets",
    "letting",
    "let's",
    "suppose",
    "supposing",
    "supposes",
    "supposed",
    "seem",
    "seeming",
    "seems",
    "seemed",
    "say",
    "saying",
    "says",
    "said",
    "know",
    "knowing",
    "knows",
    "knew",
    "known",
    "look",
    "looking",
    "looked",
    "think",
    "thinking",
    "thinks",
    "thought",
    "feel",
    "feels",
    "felt",
    "based",
    "put",
    "puts",
    #"wanted"   #Omitted since the advective is relevant
    # Question words and associated words
    "who",
    "who's",
    "who've",
    "who'd",
    "whoever",
    "whoever's",
    "whom",
    "whomever",
    "whomever's",
    "whose",
    "whosever",
```

```
    "whosever's",
    "when",
    "whenever",
    "which",
    "whichever",
    "where",
    "where's",
    "where'd",
    "wherever",
    "why",
    "why's",
    "why'd",
    "whyever",
    "what",
    "what's",
    "whatever",
    "whence",
    "how",
    "how's",
    "how'd",
    "however",
    "whether",
    "whatsoever",
    # Connector words and irrelevant adverbs
    "and",
    "or",
    "not",
    "because",
    "also",
    "always",
    "never",
    "only",
    "really",
    "very",
    "greatly",
    "extremely",
    "somewhat",
    "no",
    "nope",
    "nah",
    "yes",
    "yep",
    "yeh",
    "yeah",
    "maybe",
    "perhaps",
    "more",
```

```
    "most",
    "less",
    "least",
    "good",
    "great",
    "well",
    "better",
    "best",
    "bad",
    "worse",
    "worst",
    "too",
    "thru",
    "though",
    "although",
    "yet",
    "already",
    "then",
    "even",
    "now",
    "sometimes",
    "still",
    "together",
    "altogether",
    "entirely",
    "fully",
    "entire",
    "whole",
    "completely",
    "utterly",
    "seemingly",
    "apparently",
    "clearly",
    "obviously",
    "actually",
    "actual",
    "usually",
    "usual",
    "literally",
    "honestly",
    "absolutely",
    "definitely",
    "generally",
    "totally",
    "finally",
    "basically",
    "essentially",
```

```
    "fundamentally",
    "automatically",
    "immediately",
    "necessarily",
    "primarily",
    "normally",
    "perfectly",
    "constantly",
    "particularly",
    "eventually",
    "hopefully",
    "mainly",
    "typically",
    "specifically",
    "differently",
    "appropriately",
    "plenty",
    "certainly",
    "unfortunately",
    "ultimately",
    "unlikely",
    "likely",
    "potentially",
    "fortunately",
    "personally",
    "directly",
    "indirectly",
    "nearly",
    "closely",
    "slightly",
    "probably",
    "possibly",
    "especially",
    "frequently",
    "often",
    "oftentimes",
    "seldom",
    "rarely",
    "sure",
    "while",
    "whilst",
    "able",
    "unable",
    "else",
    "ever",
    "once",
    "twice",
```

```
    "thrice",
    "almost",
    "again",
    "instead",
    "next",
    "previous",
    "unless",
    "somehow",
    "anyhow",
    "anywhere",
    "somewhere",
    "everywhere",
    "nowhere",
    "further",
    "anymore",
    "later",
    "ago",
    "ahead",
    "just",
    "same",
    "different",
    "big",
    "small",
    "little",
    "tiny",
    "large",
    "huge",
    "pretty",
    "mostly",
    "anyway",
    "anyways",
    "otherwise",
    "regardless",
    "throughout",
    "additionally",
    "moreover",
    "furthermore",
    "meanwhile",
    "afterwards",
    # Irrelevant nouns
    "thing",
    "thing's",
    "things",
    "stuff",
    "other's",
    "others",
    "another's",
```

```
    "total",
    "",
    "false",
    "none",
    "way",
    "kind",
    # Lettered numbers and order
    "zero",
    "zeros",
    "zeroes",
    "one",
    "ones",
    "two",
    "three",
    "four",
    "five",
    "six",
    "seven",
    "eight",
    "nine",
    "ten",
    "twenty",
    "thirty",
    "forty",
    "fifty",
    "sixty",
    "seventy",
    "eighty",
    "ninety",
    "hundred",
    "hundreds",
    "thousand",
    "thousands",
    "million",
    "millions",
    "first",
    "last",
    "second",
    "third",
    "fourth",
    "fifth",
    "sixth",
    "seventh",
    "eigth",
    "ninth",
    "tenth",
    "firstly",
```

```
    "secondly",
    "thirdly",
    "lastly",
    # Greetings and slang
    "hello",
    "hi",
    "hey",
    "sup",
    "yo",
    "greetings",
    "please",
    "okay",
    "ok",
    "y'all",
    "lol",
    "rofl",
    "thank",
    "thanks",
    "alright",
    "kinda",
    "dont",
    "sorry",
    "idk",
    "tldr",
    "tl",
    "dr",  #This means that dr (doctor) is a bad feature because of tl;dr
    "tbh",
    "dude",
    "tho",
    "aka",
    "plz",
    "pls",
    "bit",
    "don",
    # Miscellaneous
    "www",
    "https",
    "http",
    "com",
    "etc",
    "html",
    "reddit",
    "subreddit",
    "subreddits",
    "comments",
    "reply",
    "replies",
```

```
    "thread",
    "threads",
    "post",
    "posts",
    "website",
    "websites",
    "web site",
    "web sites"]
print('length custom:',len(stop_words_custom))
```

```
length custom: 590
```

[ ]:
```
############################################################
```

[13]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
import numpy as np


t_start = time.time()

# Define the hyperparameters to search over
parameters = {
   'tfidf__max_df': (0.25, 0.5),
    'clf__hidden_layer_sizes': [(100,)],
    'selecter__k':[2500],
    'tfidf__ngram_range':[(1,1)],
    #'clf__learning_rate':['adaptive'],
    'clf__activation':["relu"],
        'clf__solver':["adam"],
    'clf__max_iter':[2000],
     "tfidf__stop_words": [list(stop_words_custom)],
    'clf__alpha': [0.1]
}

# Define the MLP architecture
mlp = MLPClassifier()
normalizer = Normalizer()
selecter = SelectKBest(chi2)


# Create the pipeline
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ("selecter", selecter),
```

```
    ("normalizer", normalizer),
    ('clf', mlp)
])



# Create the grid search object
grid_search = GridSearchCV(pipeline, parameters, cv=5, verbose=1, n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(train_x, train_y)


t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid_search.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid_search.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits

/usr/local/lib/python3.9/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 94.843.
The winning parameters are {'clf__activation': 'relu', 'clf__alpha': 0.1,
'clf__hidden_layer_sizes': (100,), 'clf__max_iter': 2000, 'clf__solver': 'adam',
'selecter__k': 2500, 'tfidf__max_df': 0.5, 'tfidf__ngram_range': (1, 1),
'tfidf__stop_words': ['i', "i'll", "i'd", "i'm", "i've", 'ive', 'me', 'myself',
'you', "you'll", "you'd", "you're", "you've", 'yourself', 'he', "he'll", "he'd",
"he's", 'him', 'she', "she'll", "she'd", "she's", 'her', 'it', "it'll", "it'd",
"it's", 'itself', 'oneself', 'we', "we'll", "we'd", "we're", "we've", 'us',
'ourselves', 'they', "they'll", "they'd", "they're", "they've", 'them',
'themselves', 'everyone', "everyone's", 'everybody', "everybody's", 'someone',
"someone's", 'somebody', "somebody's", 'nobody', "nobody's", 'anyone',
"anyone's", 'everything', "everything's", 'something', "something's", 'nothing',
"nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this', 'that',
"that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its', 'our',
'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots', 'some',
'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each', 'every',
'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather', 'quite',
```

'aboard', 'about', 'above', 'across', 'after', 'against', 'along', 'amid',
'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away', 'before',
'behind', 'below', 'beneath', 'beside', 'besides', 'between', 'beyond', 'but',
'by', 'concerning', 'considering', 'despite', 'down', 'during', 'except',
'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here', "here's",
'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off', 'on',
'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',

```
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etc', 'html', 'reddit',
'subreddit', 'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads',
'post', 'posts', 'website', 'websites', 'web site', 'web sites']}
Run time: 78.14167332649231 seconds
```

[14]:
```
print(round(grid_search.best_score_ * 100,3))
print(f"Run time: {elapsed_time} seconds")
y_pred = grid_search.predict(test_x)
create_test_csv(y_pred,"CNN_07032023_01.csv")
```

```
94.982
Run time: 36.68663692474365 seconds
File saved.
```

[ ]:
```
################################################################################
```

MultinomialNB

March 12, 2023

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from google.colab import drive
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import random
import time
import re
import string
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2,␣
 ↪f_classif,mutual_info_classif,f_regression
from sklearn.preprocessing import Normalizer
from sklearn import model_selection
from sklearn import svm
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[ ]:
```
True
```

[ ]:
```
#import the data
drive.mount('/content/gdrive/', force_remount=True)

train_data_initial = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/train.
 ↪csv')
test_data = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/test.csv')

print('shape train:',train_data_initial.shape)
print('shape test:',test_data.shape)
```

```
Mounted at /content/gdrive/
shape train: (718, 2)
shape test: (279, 2)
```

[ ]:
```
def shuffle_data(df):
    random.seed(0)  # Use a fixed seed for the random number generator
    df = df.sample(frac=1, random_state=0).reset_index(drop=True)
    return df
```

[ ]:
```
#function for creating the test csv file to upload to kaggle
def create_test_csv(data, outfile_name):
```

```
  rawdata= {'subreddit':data}
  csv = pd.DataFrame(rawdata, columns = ['subreddit'])
  csv.to_csv(outfile_name,index=True, header=True)
  print ("File saved.")
```

[ ]:
```
#shuffle the data and split the features from the label
train_data = shuffle_data(train_data_initial)

train_x = train_data["body"]
train_y = train_data["subreddit"]
test_x = test_data["body"]
```

[ ]:
```
#remove punctuation
def remove_punctuation(text):
   translator = str.maketrans('', '', string.punctuation)
   text = text.translate(translator)
   return text
```

[ ]:
```
print(train_x[5])
```

```
Hi there /u/LakotaPride! Welcome to /r/Trump.  [](/sp)

Thank you for posting on r/Trump Please follow all rules and guidelines. Inform
the mods if you have any concerns. [](/sp) Join our live
[discord](https://discord.gg/kh4Wv9DavE) chat to talk to your fellow patriots!
If you have any issues please reach out.


*I am a bot, and this action was performed automatically. Please [contact the
moderators of this subreddit](/message/compose/?to=/r/trump) if you have any
questions or concerns.*
```

[ ]:
```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    return text
```

[ ]:
```
def print_best_params(grid):
  bestParameters = grid.best_estimator_.get_params()
  # print(bestParameters)
  for paramName in sorted(bestParameters.keys()):
    print("\t%s: %r" % (paramName, bestParameters[paramName]))
```

[ ]:
```
#create a dictionary of stop words
stop_words_nltk = set(stopwords.words('english'))
stop_words_sklearn = text.ENGLISH_STOP_WORDS
stop_words_library = stop_words_sklearn.union(stop_words_nltk)
```

[ ]:
```
#stemmer lemmatizer
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

class LemmaTokenizer_Pos:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in␣
 ↪word_tokenize(doc) if t.isalpha()]

class LemmaTokenizer:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) if t.
 ↪isalpha()]

class LemmaTokenizer_word:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) ]


class StemTokenizer:
     def __init__(self):
       self.wnl =PorterStemmer()
     def __call__(self, doc):
       return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
```

[ ]:
```
############################################################
```

[ ]:
```
#initial training => 88.438
t_start = time.time()

pipe_params = {
}

vectorizer = CountVectorizer()
```

```
pipe = Pipeline(
    [("vect", vectorizer),("classify", MultinomialNB())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
The best accuracy is 88.438.
The winning parameters are {}
Run time: 0.5878884792327881 seconds
```

[ ]:
```
#removing punctuation  => not good
t_start = time.time()

pipe_params = {
   'vect__preprocessor': [preprocess_text,remove_punctuation,None],
    "vect__binary": [False,True]
}

vectorizer = CountVectorizer()

pipe = Pipeline(
    [("vect", vectorizer),("clf", MultinomialNB())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
```

```
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
The best accuracy is 88.438.
The winning parameters are {'vect__binary': False, 'vect__preprocessor': None}
Run time: 4.767054080963135 seconds
```

[ ]:
```
train_x_punc = train_x.copy()

for i in range(train_x.shape[0]):
  train_x_punc[i]= train_x_punc[i].translate(str.maketrans('', '', string.
 ↪punctuation))
```

[ ]:
```
#initial training,train_x_punc => worse
t_start = time.time()

pipe_params = {
}

vectorizer = CountVectorizer()

pipe = Pipeline(
    [("vect", vectorizer),("classify", MultinomialNB())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x_punc, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
The best accuracy is 87.185.
The winning parameters are {}
Run time: 0.49113988876342773 seconds
```

[ ]:
```
#stop words => stop_words_library wins 90.809.
t_start = time.time()

pipe_params = {
```

```
    "vect__binary": [False],
    "vect__stop_words":␣
 ↪[list(stop_words_nltk),list(stop_words_sklearn),list(stop_words_library),None]
}

vectorizer = CountVectorizer()

pipe = Pipeline(
    [("vect", vectorizer),("clf", MultinomialNB())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x_punc, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")


print_best_params(grid)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
The best accuracy is 90.809.
The winning parameters are {'vect__binary': False, 'vect__stop_words':
['against', 'find', "shan't", 'i', 't', 'whence', 'go', 'ten', 'she',
'somewhere', 'others', 'throughout', "don't", 'serious', 'whereafter', 'own',
'whole', 'should', 'eg', 'his', 'toward', 'whether', 'wherever', 'give', 'its',
'noone', 'is', 'were', "needn't", 'though', 'therein', 'afterwards',
'everywhere', "doesn't", 'ourselves', "you'll", 'found', 'isn', 'into',
"hadn't", 'once', 'are', 'to', 'as', 'down', 'can', 'three', 'don', "wasn't",
'twenty', 'yourselves', 'please', 'often', 'ie', 'an', 'one', 'forty', 'within',
'didn', 'side', 'mightn', 'while', 'sometime', 'hadn', 'all', 'only', "didn't",
'anyone', 'becoming', 'the', 'bottom', 'from', 'almost', 'still', 'describe',
'about', 'anyway', 'd', 'may', 'six', "that'll", 'everything', 'take', 'back',
'for', "isn't", 'mostly', 'eleven', 'whoever', 'whereas', 'moreover', 'why',
'otherwise', 'thus', "she's", 'whose', 'if', 'therefore', 'yet', 'become',
'even', 'five', 'first', 'in', 'something', 'together', 'inc', 'further',
'fill', 'elsewhere', 'very', 'whom', 'each', 'beside', 'some', 'have', 'con',
'latterly', 'themselves', 'hereupon', "it's", 'third', 'upon', 'seems', 'll',
'along', 'itself', 'indeed', 'seem', 'that', 'across', 'will', 'already',
'seemed', 'least', 'becomes', 'show', 'been', "aren't", 'couldnt', 'mill', 'it',
```

```
'except', 'because', 'nowhere', 'by', 'empty', 'out', 'but', 'after',
'beforehand', 'thereby', 'although', 'full', "haven't", 'latter', 'four',
'then', 'hence', 'her', 'see', 'could', 'you', 'these', 'none', 'thereupon',
'hereafter', 'per', 'shouldn', 'how', 'thence', 'was', 'those', 'nothing',
'perhaps', 'mustn', 'hers', 'doesn', 'there', 'nine', 'ma', 'whither', 'this',
'anyhow', 'interest', 'be', 'o', 'too', 'front', 'less', 'due', 'call',
'rather', 'just', 'without', 'name', 'everyone', 'being', 'over', 'when', 'him',
"mightn't", 's', 'amongst', 'amoungst', 'more', 'does', 'formerly', 'de', 'now',
'made', 'hundred', 'below', "you'd", 'through', 'anywhere', 'sincere', 'of',
'meanwhile', 'thin', 'behind', 'whenever', 'wasn', 'nor', 'until', 'among',
'so', 'yours', 'whereby', 'such', "shouldn't", 'sometimes', 'what', 'thru',
'much', 'same', 'must', 'again', 'a', 'am', 'off', 'never', 'ain', 'they',
'herself', 'etc', 'wouldn', 'thereafter', 'few', "you've", 'amount', 'namely',
'get', 'yourself', 'besides', 'my', 'than', 'alone', 'couldn', 'might', 'their',
'two', 'between', "won't", 'most', 'them', "weren't", 'herein', 'and', 'part',
'nevertheless', 'where', 'co', 'another', 'cant', 'bill', 'other', 'fire',
'several', 'did', 'no', 'up', 'cry', "should've", 'do', 'beyond', 'needn',
'neither', 'next', 'always', 'mine', 'put', 'wherein', 'hasn', "couldn't",
'onto', "you're", "hasn't", 'during', 'however', 'aren', 'thick', 'also', 'm',
'move', 'before', 'doing', 'un', 'which', 'with', 'keep', 'whereupon',
'anything', 'cannot', 'system', 'us', 'done', 'both', "wouldn't", 'here',
'ever', 'enough', 've', "mustn't", 'towards', 'having', 'either', 'hasnt',
'who', 'under', 'fifty', 'haven', 'fifteen', 'eight', 'me', 'former', 'he',
'hereby', 'became', 'or', 'top', 'any', 're', 'has', 'we', 'seeming', 'someone',
'ours', 'else', 'myself', 'above', 'since', 'had', 'our', 'your', 'not',
'would', 'many', 'around', 'detail', 'on', 'sixty', 'somehow', 'at', 'nobody',
'via', 'y', 'shan', 'twelve', 'theirs', 'last', 'ltd', 'every', 'himself',
'whatever', 'won', 'well', 'weren']}
Run time: 1.4538230895996094 seconds
        clf: MultinomialNB()
        clf__alpha: 1.0
        clf__class_prior: None
        clf__fit_prior: True
        clf__force_alpha: 'warn'
        memory: None
        steps: [('vect', CountVectorizer(stop_words=['against', 'find',
"shan't", 'i', 't', 'whence',
                                'go', 'ten', 'she', 'somewhere', 'others',
                                'throughout', "don't", 'serious', 'whereafter',
                                'own', 'whole', 'should', 'eg', 'his', 'toward',
                                'whether', 'wherever', 'give', 'its', 'noone', 'is',
                                'were', "needn't", 'though', …])), ('clf',
MultinomialNB())]
        vect: CountVectorizer(stop_words=['against', 'find', "shan't", 'i', 't',
'whence',
                                'go', 'ten', 'she', 'somewhere', 'others',
                                'throughout', "don't", 'serious', 'whereafter',
                                'own', 'whole', 'should', 'eg', 'his', 'toward',
```

```
                            'whether', 'wherever', 'give', 'its', 'noone', 'is',
                            'were', "needn't", 'though', …])
        vect__analyzer: 'word'
        vect__binary: False
        vect__decode_error: 'strict'
        vect__dtype: <class 'numpy.int64'>
        vect__encoding: 'utf-8'
        vect__input: 'content'
        vect__lowercase: True
        vect__max_df: 1.0
        vect__max_features: None
        vect__min_df: 1
        vect__ngram_range: (1, 1)
        vect__preprocessor: None
        vect__stop_words: ['against', 'find', "shan't", 'i', 't', 'whence',
'go', 'ten', 'she', 'somewhere', 'others', 'throughout', "don't", 'serious',
'whereafter', 'own', 'whole', 'should', 'eg', 'his', 'toward', 'whether',
'wherever', 'give', 'its', 'noone', 'is', 'were', "needn't", 'though',
'therein', 'afterwards', 'everywhere', "doesn't", 'ourselves', "you'll",
'found', 'isn', 'into', "hadn't", 'once', 'are', 'to', 'as', 'down', 'can',
'three', 'don', "wasn't", 'twenty', 'yourselves', 'please', 'often', 'ie', 'an',
'one', 'forty', 'within', 'didn', 'side', 'mightn', 'while', 'sometime', 'hadn',
'all', 'only', "didn't", 'anyone', 'becoming', 'the', 'bottom', 'from',
'almost', 'still', 'describe', 'about', 'anyway', 'd', 'may', 'six', "that'll",
'everything', 'take', 'back', 'for', "isn't", 'mostly', 'eleven', 'whoever',
'whereas', 'moreover', 'why', 'otherwise', 'thus', "she's", 'whose', 'if',
'therefore', 'yet', 'become', 'even', 'five', 'first', 'in', 'something',
'together', 'inc', 'further', 'fill', 'elsewhere', 'very', 'whom', 'each',
'beside', 'some', 'have', 'con', 'latterly', 'themselves', 'hereupon', "it's",
'third', 'upon', 'seems', 'll', 'along', 'itself', 'indeed', 'seem', 'that',
'across', 'will', 'already', 'seemed', 'least', 'becomes', 'show', 'been',
"aren't", 'couldnt', 'mill', 'it', 'except', 'because', 'nowhere', 'by',
'empty', 'out', 'but', 'after', 'beforehand', 'thereby', 'although', 'full',
"haven't", 'latter', 'four', 'then', 'hence', 'her', 'see', 'could', 'you',
'these', 'none', 'thereupon', 'hereafter', 'per', 'shouldn', 'how', 'thence',
'was', 'those', 'nothing', 'perhaps', 'mustn', 'hers', 'doesn', 'there', 'nine',
'ma', 'whither', 'this', 'anyhow', 'interest', 'be', 'o', 'too', 'front',
'less', 'due', 'call', 'rather', 'just', 'without', 'name', 'everyone', 'being',
'over', 'when', 'him', "mightn't", 's', 'amongst', 'amoungst', 'more', 'does',
'formerly', 'de', 'now', 'made', 'hundred', 'below', "you'd", 'through',
'anywhere', 'sincere', 'of', 'meanwhile', 'thin', 'behind', 'whenever', 'wasn',
'nor', 'until', 'among', 'so', 'yours', 'whereby', 'such', "shouldn't",
'sometimes', 'what', 'thru', 'much', 'same', 'must', 'again', 'a', 'am', 'off',
'never', 'ain', 'they', 'herself', 'etc', 'wouldn', 'thereafter', 'few',
"you've", 'amount', 'namely', 'get', 'yourself', 'besides', 'my', 'than',
'alone', 'couldn', 'might', 'their', 'two', 'between', "won't", 'most', 'them',
"weren't", 'herein', 'and', 'part', 'nevertheless', 'where', 'co', 'another',
'cant', 'bill', 'other', 'fire', 'several', 'did', 'no', 'up', 'cry',
```

```
"should've", 'do', 'beyond', 'needn', 'neither', 'next', 'always', 'mine',
'put', 'wherein', 'hasn', "couldn't", 'onto', "you're", "hasn't", 'during',
'however', 'aren', 'thick', 'also', 'm', 'move', 'before', 'doing', 'un',
'which', 'with', 'keep', 'whereupon', 'anything', 'cannot', 'system', 'us',
'done', 'both', "wouldn't", 'here', 'ever', 'enough', 've', "mustn't",
'towards', 'having', 'either', 'hasnt', 'who', 'under', 'fifty', 'haven',
'fifteen', 'eight', 'me', 'former', 'he', 'hereby', 'became', 'or', 'top',
'any', 're', 'has', 'we', 'seeming', 'someone', 'ours', 'else', 'myself',
'above', 'since', 'had', 'our', 'your', 'not', 'would', 'many', 'around',
'detail', 'on', 'sixty', 'somehow', 'at', 'nobody', 'via', 'y', 'shan',
'twelve', 'theirs', 'last', 'ltd', 'every', 'himself', 'whatever', 'won',
'well', 'weren']
        vect__strip_accents: None
        vect__token_pattern: '(?u)\\b\\w\\w+\\b'
        vect__tokenizer: None
        vect__vocabulary: None
        verbose: False
```

[ ]:
```python
# test alpha => 92.479. , alpha = 0.1
#selected 3
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_library)],
    "clf__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = CountVectorizer()

pipe = Pipeline([("vect", vectorizer),("clf", MultinomialNB())])

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
The best accuracy is 92.479.
The winning parameters are {'clf__alpha': 0.1, 'vect__binary': False,
```

```
'vect__stop_words': ['against', 'find', "shan't", 'i', 't', 'whence', 'go',
'ten', 'she', 'somewhere', 'others', 'throughout', "don't", 'serious',
'whereafter', 'own', 'whole', 'should', 'eg', 'his', 'toward', 'whether',
'wherever', 'give', 'its', 'noone', 'is', 'were', "needn't", 'though',
'therein', 'afterwards', 'everywhere', "doesn't", 'ourselves', "you'll",
'found', 'isn', 'into', "hadn't", 'once', 'are', 'to', 'as', 'down', 'can',
'three', 'don', "wasn't", 'twenty', 'yourselves', 'please', 'often', 'ie', 'an',
'one', 'forty', 'within', 'didn', 'side', 'mightn', 'while', 'sometime', 'hadn',
'all', 'only', "didn't", 'anyone', 'becoming', 'the', 'bottom', 'from',
'almost', 'still', 'describe', 'about', 'anyway', 'd', 'may', 'six', "that'll",
'everything', 'take', 'back', 'for', "isn't", 'mostly', 'eleven', 'whoever',
'whereas', 'moreover', 'why', 'otherwise', 'thus', "she's", 'whose', 'if',
'therefore', 'yet', 'become', 'even', 'five', 'first', 'in', 'something',
'together', 'inc', 'further', 'fill', 'elsewhere', 'very', 'whom', 'each',
'beside', 'some', 'have', 'con', 'latterly', 'themselves', 'hereupon', "it's",
'third', 'upon', 'seems', 'll', 'along', 'itself', 'indeed', 'seem', 'that',
'across', 'will', 'already', 'seemed', 'least', 'becomes', 'show', 'been',
"aren't", 'couldnt', 'mill', 'it', 'except', 'because', 'nowhere', 'by',
'empty', 'out', 'but', 'after', 'beforehand', 'thereby', 'although', 'full',
"haven't", 'latter', 'four', 'then', 'hence', 'her', 'see', 'could', 'you',
'these', 'none', 'thereupon', 'hereafter', 'per', 'shouldn', 'how', 'thence',
'was', 'those', 'nothing', 'perhaps', 'mustn', 'hers', 'doesn', 'there', 'nine',
'ma', 'whither', 'this', 'anyhow', 'interest', 'be', 'o', 'too', 'front',
'less', 'due', 'call', 'rather', 'just', 'without', 'name', 'everyone', 'being',
'over', 'when', 'him', "mightn't", 's', 'amongst', 'amoungst', 'more', 'does',
'formerly', 'de', 'now', 'made', 'hundred', 'below', "you'd", 'through',
'anywhere', 'sincere', 'of', 'meanwhile', 'thin', 'behind', 'whenever', 'wasn',
'nor', 'until', 'among', 'so', 'yours', 'whereby', 'such', "shouldn't",
'sometimes', 'what', 'thru', 'much', 'same', 'must', 'again', 'a', 'am', 'off',
'never', 'ain', 'they', 'herself', 'etc', 'wouldn', 'thereafter', 'few',
"you've", 'amount', 'namely', 'get', 'yourself', 'besides', 'my', 'than',
'alone', 'couldn', 'might', 'their', 'two', 'between', "won't", 'most', 'them',
"weren't", 'herein', 'and', 'part', 'nevertheless', 'where', 'co', 'another',
'cant', 'bill', 'other', 'fire', 'several', 'did', 'no', 'up', 'cry',
"should've", 'do', 'beyond', 'needn', 'neither', 'next', 'always', 'mine',
'put', 'wherein', 'hasn', "couldn't", 'onto', "you're", "hasn't", 'during',
'however', 'aren', 'thick', 'also', 'm', 'move', 'before', 'doing', 'un',
'which', 'with', 'keep', 'whereupon', 'anything', 'cannot', 'system', 'us',
'done', 'both', "wouldn't", 'here', 'ever', 'enough', 've', "mustn't",
'towards', 'having', 'either', 'hasnt', 'who', 'under', 'fifty', 'haven',
'fifteen', 'eight', 'me', 'former', 'he', 'hereby', 'became', 'or', 'top',
'any', 're', 'has', 'we', 'seeming', 'someone', 'ours', 'else', 'myself',
'above', 'since', 'had', 'our', 'your', 'not', 'would', 'many', 'around',
'detail', 'on', 'sixty', 'somehow', 'at', 'nobody', 'via', 'y', 'shan',
'twelve', 'theirs', 'last', 'ltd', 'every', 'himself', 'whatever', 'won',
'well', 'weren']}
Run time: 2.2182962894439697 seconds
```

[ ]:
```
# test selector = > decreased(90.669.)
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_library)],
    "clf__alpha" : [0.001, 0.01, 0.1,0.02,0.5],
    "selecter__k":[5000,3000,6000]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)

pipe = Pipeline([("vect", vectorizer),("selecter", selecter),("clf",␣
 ↪MultinomialNB())])

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x_punc, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
The best accuracy is 90.669.
The winning parameters are {'clf__alpha': 0.5, 'selecter__k': 5000,
'vect__binary': False, 'vect__stop_words': ['against', 'find', "shan't", 'i',
't', 'whence', 'go', 'ten', 'she', 'somewhere', 'others', 'throughout', "don't",
'serious', 'whereafter', 'own', 'whole', 'should', 'eg', 'his', 'toward',
'whether', 'wherever', 'give', 'its', 'noone', 'is', 'were', "needn't",
'though', 'therein', 'afterwards', 'everywhere', "doesn't", 'ourselves',
"you'll", 'found', 'isn', 'into', "hadn't", 'once', 'are', 'to', 'as', 'down',
'can', 'three', 'don', "wasn't", 'twenty', 'yourselves', 'please', 'often',
'ie', 'an', 'one', 'forty', 'within', 'didn', 'side', 'mightn', 'while',
'sometime', 'hadn', 'all', 'only', "didn't", 'anyone', 'becoming', 'the',
'bottom', 'from', 'almost', 'still', 'describe', 'about', 'anyway', 'd', 'may',
'six', "that'll", 'everything', 'take', 'back', 'for', "isn't", 'mostly',
'eleven', 'whoever', 'whereas', 'moreover', 'why', 'otherwise', 'thus', "she's",
'whose', 'if', 'therefore', 'yet', 'become', 'even', 'five', 'first', 'in',
'something', 'together', 'inc', 'further', 'fill', 'elsewhere', 'very', 'whom',
'each', 'beside', 'some', 'have', 'con', 'latterly', 'themselves', 'hereupon',
```

```
"it's", 'third', 'upon', 'seems', 'll', 'along', 'itself', 'indeed', 'seem',
'that', 'across', 'will', 'already', 'seemed', 'least', 'becomes', 'show',
'been', "aren't", 'couldnt', 'mill', 'it', 'except', 'because', 'nowhere', 'by',
'empty', 'out', 'but', 'after', 'beforehand', 'thereby', 'although', 'full',
"haven't", 'latter', 'four', 'then', 'hence', 'her', 'see', 'could', 'you',
'these', 'none', 'thereupon', 'hereafter', 'per', 'shouldn', 'how', 'thence',
'was', 'those', 'nothing', 'perhaps', 'mustn', 'hers', 'doesn', 'there', 'nine',
'ma', 'whither', 'this', 'anyhow', 'interest', 'be', 'o', 'too', 'front',
'less', 'due', 'call', 'rather', 'just', 'without', 'name', 'everyone', 'being',
'over', 'when', 'him', "mightn't", 's', 'amongst', 'amoungst', 'more', 'does',
'formerly', 'de', 'now', 'made', 'hundred', 'below', "you'd", 'through',
'anywhere', 'sincere', 'of', 'meanwhile', 'thin', 'behind', 'whenever', 'wasn',
'nor', 'until', 'among', 'so', 'yours', 'whereby', 'such', "shouldn't",
'sometimes', 'what', 'thru', 'much', 'same', 'must', 'again', 'a', 'am', 'off',
'never', 'ain', 'they', 'herself', 'etc', 'wouldn', 'thereafter', 'few',
"you've", 'amount', 'namely', 'get', 'yourself', 'besides', 'my', 'than',
'alone', 'couldn', 'might', 'their', 'two', 'between', "won't", 'most', 'them',
"weren't", 'herein', 'and', 'part', 'nevertheless', 'where', 'co', 'another',
'cant', 'bill', 'other', 'fire', 'several', 'did', 'no', 'up', 'cry',
"should've", 'do', 'beyond', 'needn', 'neither', 'next', 'always', 'mine',
'put', 'wherein', 'hasn', "couldn't", 'onto', "you're", "hasn't", 'during',
'however', 'aren', 'thick', 'also', 'm', 'move', 'before', 'doing', 'un',
'which', 'with', 'keep', 'whereupon', 'anything', 'cannot', 'system', 'us',
'done', 'both', "wouldn't", 'here', 'ever', 'enough', 've', "mustn't",
'towards', 'having', 'either', 'hasnt', 'who', 'under', 'fifty', 'haven',
'fifteen', 'eight', 'me', 'former', 'he', 'hereby', 'became', 'or', 'top',
'any', 're', 'has', 'we', 'seeming', 'someone', 'ours', 'else', 'myself',
'above', 'since', 'had', 'our', 'your', 'not', 'would', 'many', 'around',
'detail', 'on', 'sixty', 'somehow', 'at', 'nobody', 'via', 'y', 'shan',
'twelve', 'theirs', 'last', 'ltd', 'every', 'himself', 'whatever', 'won',
'well', 'weren']}
Run time: 12.57723093032837 seconds
```

[ ]:
```
#testing normalizer , without : 92.479, with:92.199., 92.34. with norm max =>␣
 ↪no normalizer
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_library)],
    "clf__alpha" : [0.001, 0.01, 0.1,0.02,0.5],
    'normalizer__norm': ['l1','l2','max']
}

vectorizer = CountVectorizer()
normalizer = Normalizer()
```

```
pipe = Pipeline([("vect", vectorizer),("normalizer", normalizer),("clf",␣
 ↪MultinomialNB())])
#pipe = Pipeline([("vect", vectorizer),("clf", MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
The best accuracy is 91.779.
The winning parameters are {'clf__alpha': 0.01, 'normalizer__norm': 'l1',
'vect__binary': False, 'vect__stop_words': ['thick', 'thru', 'cant', 'below',
'ma', 'becomes', "you've", 'thus', 'fire', 'somewhere', 'latter', 'after',
'much', 'put', 'sometimes', 's', 'see', "aren't", 'seem', 'interest', 'if',
'elsewhere', 'over', 'less', "won't", 'ours', 'ain', 't', 'under', 'anyway',
'whoever', 'ourselves', 'hence', 'not', 'd', 'become', 've', 'should', 'no',
'toward', 'i', 'besides', 'therein', 'something', 'beforehand', 'out', "shan't",
'or', 'through', 'why', 'inc', 'upon', 'last', 'few', 'perhaps', 'one', 'found',
'themselves', 'find', 'again', 'now', 'while', 'same', 'doesn', 'who', 'with',
'formerly', 'eg', 'already', 'side', 'isn', 'don', 'which', "you're", 'give',
'is', 'however', 'couldnt', 'indeed', 'cry', 'nine', 'in', 'someone', 'many',
'whereby', 'before', 'further', 'the', 'whereas', 'often', 'amoungst',
'latterly', "shouldn't", 'they', 'meanwhile', 'our', 'twenty', 'herself',
'once', 'always', 'done', 'namely', 'against', 'wherein', 'still', 'wasn',
'etc', 'his', 'though', 'other', 'all', 'up', 'get', 'herein', 'can', "weren't",
'others', 'because', 'along', 'whole', 'former', 'its', "mightn't", 'keep',
'has', 'as', 'how', 'these', 'shouldn', 'me', 'wouldn', 'what', "you'll",
'seemed', 'within', 'those', 'hasn', 'every', 'hasnt', 'hundred', 'since', 'of',
'didn', "she's", 'via', 'here', 'per', 'otherwise', 'wherever', "you'd",
'whereupon', 'haven', 'never', 'anything', 'empty', 'seems', 'might', 'just',
'next', 'ltd', 'to', 'y', 'couldn', 'hadn', 'by', 'nowhere', 'among', "mustn't",
'seeming', 'it', 'call', 'theirs', 'each', 'behind', 'everything', 'amount',
'de', 'down', 'did', 'alone', "don't", 'about', 'sometime', 'an', 'also',
'will', 'weren', "doesn't", 'sincere', 'whither', 'whenever', 'thence',
'mostly', 'hereby', 'serious', 'twelve', 'doing', 'bill', 'ie', 'made',
'together', 'when', 'eight', 'thereafter', 'third', 'am', 'well', 'll',
```

```
'detail', "couldn't", 'either', 'won', 'where', 'very', 'been', 'she', 'was',
'this', 'front', 'therefore', 'sixty', 'whence', 'beyond', 'were', 'several',
'amongst', 'o', 'three', 'throughout', 're', 'into', 'he', 'shan', 'mustn',
'needn', 'own', 'do', 'anyone', 'first', 'almost', 'due', 'system', 'than',
'con', 'fifteen', 'eleven', 'enough', "needn't", 'mightn', 'most', 'more',
'are', 'everywhere', 'thin', 'that', 'yourselves', 'them', 'fill', 'nothing',
'having', 'at', "didn't", 'may', 'on', 'top', 'became', 'you', 'any', 'take',
'their', 'during', 'only', 'neither', 'whatever', 'us', 'none', 'have', 'both',
'hereupon', 'five', 'cannot', 'mill', 'although', 'co', 'from', 'somehow',
'moreover', 'onto', 'm', 'nevertheless', 'some', 'please', 'too', 'and',
'except', 'even', 'go', 'himself', 'yourself', 'hers', 'bottom', 'un',
'whether', 'another', 'around', "haven't", 'nor', 'such', "hasn't", 'beside',
'whose', 'then', 'two', 'being', 'aren', 'had', 'full', 'whom', 'ten',
'hereafter', 'could', 'there', 'else', 'rather', 'him', 'itself', 'her', 'your',
'thereupon', 'my', 'mine', 'move', 'but', 'ever', 'describe', 'show',
'afterwards', 'noone', 'six', 'thereby', 'we', 'be', "isn't", 'name', 'would',
'a', "it's", 'anywhere', 'anyhow', 'for', 'towards', "wasn't", 'so', 'off',
'yours', 'four', 'without', 'becoming', 'whereafter', "that'll", 'across',
'everyone', 'fifty', 'myself', 'yet', 'until', 'part', 'least', 'nobody',
'must', 'between', "should've", 'above', "hadn't", "wouldn't", 'back', 'does',
'forty']}
Run time: 2.6477572917938232 seconds
```

[ ]:
```
#testing lemma,stemmizer => not working
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_library)],
    "vect__tokenizer": [LemmaTokenizer_word()],
    'selecter__k':[5000,3000],
    "clf__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)


pipe = Pipeline([("vect", vectorizer),("selecter", selecter),("clf",␣
 ↪MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()
```

```
elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:528:
UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is
not None'
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'ll", "'re", "'s", "'ve",
'make', "n't", 'need', 'sha', 'win', 'wo'] not in stop_words.
  warnings.warn(
```

```
The best accuracy is 89.28.
The winning parameters are {'clf__alpha': 0.001, 'selecter__k': 3000,
'vect__binary': False, 'vect__stop_words': ['thick', 'thru', 'cant', 'below',
'ma', 'becomes', "you've", 'thus', 'fire', 'somewhere', 'latter', 'after',
'much', 'put', 'sometimes', 's', 'see', "aren't", 'seem', 'interest', 'if',
'elsewhere', 'over', 'less', "won't", 'ours', 'ain', 't', 'under', 'anyway',
'whoever', 'ourselves', 'hence', 'not', 'd', 'become', 've', 'should', 'no',
'toward', 'i', 'besides', 'therein', 'something', 'beforehand', 'out', "shan't",
'or', 'through', 'why', 'inc', 'upon', 'last', 'few', 'perhaps', 'one', 'found',
'themselves', 'find', 'again', 'now', 'while', 'same', 'doesn', 'who', 'with',
'formerly', 'eg', 'already', 'side', 'isn', 'don', 'which', "you're", 'give',
'is', 'however', 'couldnt', 'indeed', 'cry', 'nine', 'in', 'someone', 'many',
'whereby', 'before', 'further', 'the', 'whereas', 'often', 'amoungst',
'latterly', "shouldn't", 'they', 'meanwhile', 'our', 'twenty', 'herself',
'once', 'always', 'done', 'namely', 'against', 'wherein', 'still', 'wasn',
'etc', 'his', 'though', 'other', 'all', 'up', 'get', 'herein', 'can', "weren't",
'others', 'because', 'along', 'whole', 'former', 'its', "mightn't", 'keep',
'has', 'as', 'how', 'these', 'shouldn', 'me', 'wouldn', 'what', "you'll",
'seemed', 'within', 'those', 'hasn', 'every', 'hasnt', 'hundred', 'since', 'of',
'didn', "she's", 'via', 'here', 'per', 'otherwise', 'wherever', "you'd",
'whereupon', 'haven', 'never', 'anything', 'empty', 'seems', 'might', 'just',
'next', 'ltd', 'to', 'y', 'couldn', 'hadn', 'by', 'nowhere', 'among', "mustn't",
'seeming', 'it', 'call', 'theirs', 'each', 'behind', 'everything', 'amount',
'de', 'down', 'did', 'alone', "don't", 'about', 'sometime', 'an', 'also',
'will', 'weren', "doesn't", 'sincere', 'whither', 'whenever', 'thence',
'mostly', 'hereby', 'serious', 'twelve', 'doing', 'bill', 'ie', 'made',
'together', 'when', 'eight', 'thereafter', 'third', 'am', 'well', 'll',
'detail', "couldn't", 'either', 'won', 'where', 'very', 'been', 'she', 'was',
```

```
'this', 'front', 'therefore', 'sixty', 'whence', 'beyond', 'were', 'several',
'amongst', 'o', 'three', 'throughout', 're', 'into', 'he', 'shan', 'mustn',
'needn', 'own', 'do', 'anyone', 'first', 'almost', 'due', 'system', 'than',
'con', 'fifteen', 'eleven', 'enough', "needn't", 'mightn', 'most', 'more',
'are', 'everywhere', 'thin', 'that', 'yourselves', 'them', 'fill', 'nothing',
'having', 'at', "didn't", 'may', 'on', 'top', 'became', 'you', 'any', 'take',
'their', 'during', 'only', 'neither', 'whatever', 'us', 'none', 'have', 'both',
'hereupon', 'five', 'cannot', 'mill', 'although', 'co', 'from', 'somehow',
'moreover', 'onto', 'm', 'nevertheless', 'some', 'please', 'too', 'and',
'except', 'even', 'go', 'himself', 'yourself', 'hers', 'bottom', 'un',
'whether', 'another', 'around', "haven't", 'nor', 'such', "hasn't", 'beside',
'whose', 'then', 'two', 'being', 'aren', 'had', 'full', 'whom', 'ten',
'hereafter', 'could', 'there', 'else', 'rather', 'him', 'itself', 'her', 'your',
'thereupon', 'my', 'mine', 'move', 'but', 'ever', 'describe', 'show',
'afterwards', 'noone', 'six', 'thereby', 'we', 'be', "isn't", 'name', 'would',
'a', "it's", 'anywhere', 'anyhow', 'for', 'towards', "wasn't", 'so', 'off',
'yours', 'four', 'without', 'becoming', 'whereafter', "that'll", 'across',
'everyone', 'fifty', 'myself', 'yet', 'until', 'part', 'least', 'nobody',
'must', 'between', "should've", 'above', "hadn't", "wouldn't", 'back', 'does',
'forty'], 'vect__tokenizer': <__main__.LemmaTokenizer_word object at
0x7f4c16e05460>}
Run time: 66.79015469551086 seconds
```

[ ]:
```
#test ngram() ,best is 92.47 , 93.176. with ngram(1,2)
#selected 1
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_library)],
    'vect__ngram_range':[(1,1),(1,2),(1,3)],
    "clf__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = CountVectorizer()


pipe = Pipeline([("vect", vectorizer),("clf", MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
```

```
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
The best accuracy is 93.176.
The winning parameters are {'clf__alpha': 0.5, 'vect__binary': False,
'vect__ngram_range': (1, 2), 'vect__stop_words': ['against', 'find', "shan't",
'i', 't', 'whence', 'go', 'ten', 'she', 'somewhere', 'others', 'throughout',
"don't", 'serious', 'whereafter', 'own', 'whole', 'should', 'eg', 'his',
'toward', 'whether', 'wherever', 'give', 'its', 'noone', 'is', 'were',
"needn't", 'though', 'therein', 'afterwards', 'everywhere', "doesn't",
'ourselves', "you'll", 'found', 'isn', 'into', "hadn't", 'once', 'are', 'to',
'as', 'down', 'can', 'three', 'don', "wasn't", 'twenty', 'yourselves', 'please',
'often', 'ie', 'an', 'one', 'forty', 'within', 'didn', 'side', 'mightn',
'while', 'sometime', 'hadn', 'all', 'only', "didn't", 'anyone', 'becoming',
'the', 'bottom', 'from', 'almost', 'still', 'describe', 'about', 'anyway', 'd',
'may', 'six', "that'll", 'everything', 'take', 'back', 'for', "isn't", 'mostly',
'eleven', 'whoever', 'whereas', 'moreover', 'why', 'otherwise', 'thus', "she's",
'whose', 'if', 'therefore', 'yet', 'become', 'even', 'five', 'first', 'in',
'something', 'together', 'inc', 'further', 'fill', 'elsewhere', 'very', 'whom',
'each', 'beside', 'some', 'have', 'con', 'latterly', 'themselves', 'hereupon',
"it's", 'third', 'upon', 'seems', 'll', 'along', 'itself', 'indeed', 'seem',
'that', 'across', 'will', 'already', 'seemed', 'least', 'becomes', 'show',
'been', "aren't", 'couldnt', 'mill', 'it', 'except', 'because', 'nowhere', 'by',
'empty', 'out', 'but', 'after', 'beforehand', 'thereby', 'although', 'full',
"haven't", 'latter', 'four', 'then', 'hence', 'her', 'see', 'could', 'you',
'these', 'none', 'thereupon', 'hereafter', 'per', 'shouldn', 'how', 'thence',
'was', 'those', 'nothing', 'perhaps', 'mustn', 'hers', 'doesn', 'there', 'nine',
'ma', 'whither', 'this', 'anyhow', 'interest', 'be', 'o', 'too', 'front',
'less', 'due', 'call', 'rather', 'just', 'without', 'name', 'everyone', 'being',
'over', 'when', 'him', "mightn't", 's', 'amongst', 'amoungst', 'more', 'does',
'formerly', 'de', 'now', 'made', 'hundred', 'below', "you'd", 'through',
'anywhere', 'sincere', 'of', 'meanwhile', 'thin', 'behind', 'whenever', 'wasn',
'nor', 'until', 'among', 'so', 'yours', 'whereby', 'such', "shouldn't",
'sometimes', 'what', 'thru', 'much', 'same', 'must', 'again', 'a', 'am', 'off',
'never', 'ain', 'they', 'herself', 'etc', 'wouldn', 'thereafter', 'few',
"you've", 'amount', 'namely', 'get', 'yourself', 'besides', 'my', 'than',
'alone', 'couldn', 'might', 'their', 'two', 'between', "won't", 'most', 'them',
"weren't", 'herein', 'and', 'part', 'nevertheless', 'where', 'co', 'another',
'cant', 'bill', 'other', 'fire', 'several', 'did', 'no', 'up', 'cry',
"should've", 'do', 'beyond', 'needn', 'neither', 'next', 'always', 'mine',
'put', 'wherein', 'hasn', "couldn't", 'onto', "you're", "hasn't", 'during',
'however', 'aren', 'thick', 'also', 'm', 'move', 'before', 'doing', 'un',
'which', 'with', 'keep', 'whereupon', 'anything', 'cannot', 'system', 'us',
```

```
'done', 'both', "wouldn't", 'here', 'ever', 'enough', 've', "mustn't",
'towards', 'having', 'either', 'hasnt', 'who', 'under', 'fifty', 'haven',
'fifteen', 'eight', 'me', 'former', 'he', 'hereby', 'became', 'or', 'top',
'any', 're', 'has', 'we', 'seeming', 'someone', 'ours', 'else', 'myself',
'above', 'since', 'had', 'our', 'your', 'not', 'would', 'many', 'around',
'detail', 'on', 'sixty', 'somehow', 'at', 'nobody', 'via', 'y', 'shan',
'twelve', 'theirs', 'last', 'ltd', 'every', 'himself', 'whatever', 'won',
'well', 'weren']}
Run time: 17.149862051010132 seconds
```

[ ]:
```
#test CountVectorizer =>93.176
#TfidfVectorizer with (1,1) ngram and selector chi2 =>92.058.
#selected 2
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_library)],
    'vect__ngram_range':[(1,1)],
    "clf__alpha" : [0.01, 0.1,0.02,0.5],
    'selecter__k':[5000,3000]
}

vectorizer = TfidfVectorizer()
normalizer = Normalizer()
selecter = SelectKBest(chi2)



pipe = Pipeline([("vect", vectorizer),("normalizer", normalizer),("selecter",␣
 ↪selecter),("clf", MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
The best accuracy is 92.058.
```

```
The winning parameters are {'clf__alpha': 0.01, 'selecter__k': 5000,
'vect__binary': False, 'vect__ngram_range': (1, 1), 'vect__stop_words':
['against', 'find', "shan't", 'i', 't', 'whence', 'go', 'ten', 'she',
'somewhere', 'others', 'throughout', "don't", 'serious', 'whereafter', 'own',
'whole', 'should', 'eg', 'his', 'toward', 'whether', 'wherever', 'give', 'its',
'noone', 'is', 'were', "needn't", 'though', 'therein', 'afterwards',
'everywhere', "doesn't", 'ourselves', "you'll", 'found', 'isn', 'into',
"hadn't", 'once', 'are', 'to', 'as', 'down', 'can', 'three', 'don', "wasn't",
'twenty', 'yourselves', 'please', 'often', 'ie', 'an', 'one', 'forty', 'within',
'didn', 'side', 'mightn', 'while', 'sometime', 'hadn', 'all', 'only', "didn't",
'anyone', 'becoming', 'the', 'bottom', 'from', 'almost', 'still', 'describe',
'about', 'anyway', 'd', 'may', 'six', "that'll", 'everything', 'take', 'back',
'for', "isn't", 'mostly', 'eleven', 'whoever', 'whereas', 'moreover', 'why',
'otherwise', 'thus', "she's", 'whose', 'if', 'therefore', 'yet', 'become',
'even', 'five', 'first', 'in', 'something', 'together', 'inc', 'further',
'fill', 'elsewhere', 'very', 'whom', 'each', 'beside', 'some', 'have', 'con',
'latterly', 'themselves', 'hereupon', "it's", 'third', 'upon', 'seems', 'll',
'along', 'itself', 'indeed', 'seem', 'that', 'across', 'will', 'already',
'seemed', 'least', 'becomes', 'show', 'been', "aren't", 'couldnt', 'mill', 'it',
'except', 'because', 'nowhere', 'by', 'empty', 'out', 'but', 'after',
'beforehand', 'thereby', 'although', 'full', "haven't", 'latter', 'four',
'then', 'hence', 'her', 'see', 'could', 'you', 'these', 'none', 'thereupon',
'hereafter', 'per', 'shouldn', 'how', 'thence', 'was', 'those', 'nothing',
'perhaps', 'mustn', 'hers', 'doesn', 'there', 'nine', 'ma', 'whither', 'this',
'anyhow', 'interest', 'be', 'o', 'too', 'front', 'less', 'due', 'call',
'rather', 'just', 'without', 'name', 'everyone', 'being', 'over', 'when', 'him',
"mightn't", 's', 'amongst', 'amoungst', 'more', 'does', 'formerly', 'de', 'now',
'made', 'hundred', 'below', "you'd", 'through', 'anywhere', 'sincere', 'of',
'meanwhile', 'thin', 'behind', 'whenever', 'wasn', 'nor', 'until', 'among',
'so', 'yours', 'whereby', 'such', "shouldn't", 'sometimes', 'what', 'thru',
'much', 'same', 'must', 'again', 'a', 'am', 'off', 'never', 'ain', 'they',
'herself', 'etc', 'wouldn', 'thereafter', 'few', "you've", 'amount', 'namely',
'get', 'yourself', 'besides', 'my', 'than', 'alone', 'couldn', 'might', 'their',
'two', 'between', "won't", 'most', 'them', "weren't", 'herein', 'and', 'part',
'nevertheless', 'where', 'co', 'another', 'cant', 'bill', 'other', 'fire',
'several', 'did', 'no', 'up', 'cry', "should've", 'do', 'beyond', 'needn',
'neither', 'next', 'always', 'mine', 'put', 'wherein', 'hasn', "couldn't",
'onto', "you're", "hasn't", 'during', 'however', 'aren', 'thick', 'also', 'm',
'move', 'before', 'doing', 'un', 'which', 'with', 'keep', 'whereupon',
'anything', 'cannot', 'system', 'us', 'done', 'both', "wouldn't", 'here',
'ever', 'enough', 've', "mustn't", 'towards', 'having', 'either', 'hasnt',
'who', 'under', 'fifty', 'haven', 'fifteen', 'eight', 'me', 'former', 'he',
'hereby', 'became', 'or', 'top', 'any', 're', 'has', 'we', 'seeming', 'someone',
'ours', 'else', 'myself', 'above', 'since', 'had', 'our', 'your', 'not',
'would', 'many', 'around', 'detail', 'on', 'sixty', 'somehow', 'at', 'nobody',
'via', 'y', 'shan', 'twelve', 'theirs', 'last', 'ltd', 'every', 'himself',
'whatever', 'won', 'well', 'weren']}
Run time: 9.161921262741089 seconds
```

[ ]:
```
#confirm 93.1
#same as selected 1
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_library)],
    'vect__ngram_range':[(1,2)],
    "clf__alpha" : [0.5]
}

vectorizer = CountVectorizer()



pipe = Pipeline([("vect", vectorizer),("clf", MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
The best accuracy is 93.176.
The winning parameters are {'clf__alpha': 0.5, 'vect__binary': False,
'vect__ngram_range': (1, 2), 'vect__stop_words': ['against', 'find', "shan't",
'i', 't', 'whence', 'go', 'ten', 'she', 'somewhere', 'others', 'throughout',
"don't", 'serious', 'whereafter', 'own', 'whole', 'should', 'eg', 'his',
'toward', 'whether', 'wherever', 'give', 'its', 'noone', 'is', 'were',
"needn't", 'though', 'therein', 'afterwards', 'everywhere', "doesn't",
'ourselves', "you'll", 'found', 'isn', 'into', "hadn't", 'once', 'are', 'to',
'as', 'down', 'can', 'three', 'don', "wasn't", 'twenty', 'yourselves', 'please',
'often', 'ie', 'an', 'one', 'forty', 'within', 'didn', 'side', 'mightn',
'while', 'sometime', 'hadn', 'all', 'only', "didn't", 'anyone', 'becoming',
'the', 'bottom', 'from', 'almost', 'still', 'describe', 'about', 'anyway', 'd',
'may', 'six', "that'll", 'everything', 'take', 'back', 'for', "isn't", 'mostly',
'eleven', 'whoever', 'whereas', 'moreover', 'why', 'otherwise', 'thus', "she's",
'whose', 'if', 'therefore', 'yet', 'become', 'even', 'five', 'first', 'in',
```

```
'something', 'together', 'inc', 'further', 'fill', 'elsewhere', 'very', 'whom',
'each', 'beside', 'some', 'have', 'con', 'latterly', 'themselves', 'hereupon',
"it's", 'third', 'upon', 'seems', 'll', 'along', 'itself', 'indeed', 'seem',
'that', 'across', 'will', 'already', 'seemed', 'least', 'becomes', 'show',
'been', "aren't", 'couldnt', 'mill', 'it', 'except', 'because', 'nowhere', 'by',
'empty', 'out', 'but', 'after', 'beforehand', 'thereby', 'although', 'full',
"haven't", 'latter', 'four', 'then', 'hence', 'her', 'see', 'could', 'you',
'these', 'none', 'thereupon', 'hereafter', 'per', 'shouldn', 'how', 'thence',
'was', 'those', 'nothing', 'perhaps', 'mustn', 'hers', 'doesn', 'there', 'nine',
'ma', 'whither', 'this', 'anyhow', 'interest', 'be', 'o', 'too', 'front',
'less', 'due', 'call', 'rather', 'just', 'without', 'name', 'everyone', 'being',
'over', 'when', 'him', "mightn't", 's', 'amongst', 'amoungst', 'more', 'does',
'formerly', 'de', 'now', 'made', 'hundred', 'below', "you'd", 'through',
'anywhere', 'sincere', 'of', 'meanwhile', 'thin', 'behind', 'whenever', 'wasn',
'nor', 'until', 'among', 'so', 'yours', 'whereby', 'such', "shouldn't",
'sometimes', 'what', 'thru', 'much', 'same', 'must', 'again', 'a', 'am', 'off',
'never', 'ain', 'they', 'herself', 'etc', 'wouldn', 'thereafter', 'few',
"you've", 'amount', 'namely', 'get', 'yourself', 'besides', 'my', 'than',
'alone', 'couldn', 'might', 'their', 'two', 'between', "won't", 'most', 'them',
"weren't", 'herein', 'and', 'part', 'nevertheless', 'where', 'co', 'another',
'cant', 'bill', 'other', 'fire', 'several', 'did', 'no', 'up', 'cry',
"should've", 'do', 'beyond', 'needn', 'neither', 'next', 'always', 'mine',
'put', 'wherein', 'hasn', "couldn't", 'onto', "you're", "hasn't", 'during',
'however', 'aren', 'thick', 'also', 'm', 'move', 'before', 'doing', 'un',
'which', 'with', 'keep', 'whereupon', 'anything', 'cannot', 'system', 'us',
'done', 'both', "wouldn't", 'here', 'ever', 'enough', 've', "mustn't",
'towards', 'having', 'either', 'hasnt', 'who', 'under', 'fifty', 'haven',
'fifteen', 'eight', 'me', 'former', 'he', 'hereby', 'became', 'or', 'top',
'any', 're', 'has', 'we', 'seeming', 'someone', 'ours', 'else', 'myself',
'above', 'since', 'had', 'our', 'your', 'not', 'would', 'many', 'around',
'detail', 'on', 'sixty', 'somehow', 'at', 'nobody', 'via', 'y', 'shan',
'twelve', 'theirs', 'last', 'ltd', 'every', 'himself', 'whatever', 'won',
'well', 'weren']}
Run time: 1.3021674156188965 seconds
```

[ ]:
```python
#test selector
#[chi2, f_classif, mutual_info_classif, f_regression, mutual_info_regression]
#fclassic : 91.225. chi2: 91.084
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_library)],
    'vect__ngram_range':[(1,2)],
    "selecter__score_func": [mutual_info_classif],
    "selecter__k":[5000,3000],
    "clf__alpha" : [0.01, 0.1,0.02,0.5]
```

```
}

vectorizer = CountVectorizer()
selecter = SelectKBest()


pipe = Pipeline([("vect", vectorizer),("selecter", selecter),("clf",␣
 ↪MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
The best accuracy is 90.807.
The winning parameters are {'clf__alpha': 0.02, 'selecter__k': 5000,
'selecter__score_func': <function mutual_info_classif at 0x7f4c2cdaf550>,
'vect__binary': False, 'vect__ngram_range': (1, 2), 'vect__stop_words':
['thick', 'thru', 'cant', 'below', 'ma', 'becomes', "you've", 'thus', 'fire',
'somewhere', 'latter', 'after', 'much', 'put', 'sometimes', 's', 'see',
"aren't", 'seem', 'interest', 'if', 'elsewhere', 'over', 'less', "won't",
'ours', 'ain', 't', 'under', 'anyway', 'whoever', 'ourselves', 'hence', 'not',
'd', 'become', 've', 'should', 'no', 'toward', 'i', 'besides', 'therein',
'something', 'beforehand', 'out', "shan't", 'or', 'through', 'why', 'inc',
'upon', 'last', 'few', 'perhaps', 'one', 'found', 'themselves', 'find', 'again',
'now', 'while', 'same', 'doesn', 'who', 'with', 'formerly', 'eg', 'already',
'side', 'isn', 'don', 'which', "you're", 'give', 'is', 'however', 'couldnt',
'indeed', 'cry', 'nine', 'in', 'someone', 'many', 'whereby', 'before',
'further', 'the', 'whereas', 'often', 'amoungst', 'latterly', "shouldn't",
'they', 'meanwhile', 'our', 'twenty', 'herself', 'once', 'always', 'done',
'namely', 'against', 'wherein', 'still', 'wasn', 'etc', 'his', 'though',
'other', 'all', 'up', 'get', 'herein', 'can', "weren't", 'others', 'because',
'along', 'whole', 'former', 'its', "mightn't", 'keep', 'has', 'as', 'how',
'these', 'shouldn', 'me', 'wouldn', 'what', "you'll", 'seemed', 'within',
'those', 'hasn', 'every', 'hasnt', 'hundred', 'since', 'of', 'didn', "she's",
'via', 'here', 'per', 'otherwise', 'wherever', "you'd", 'whereupon', 'haven',
'never', 'anything', 'empty', 'seems', 'might', 'just', 'next', 'ltd', 'to',
```

```
'y', 'couldn', 'hadn', 'by', 'nowhere', 'among', "mustn't", 'seeming', 'it',
'call', 'theirs', 'each', 'behind', 'everything', 'amount', 'de', 'down', 'did',
'alone', "don't", 'about', 'sometime', 'an', 'also', 'will', 'weren', "doesn't",
'sincere', 'whither', 'whenever', 'thence', 'mostly', 'hereby', 'serious',
'twelve', 'doing', 'bill', 'ie', 'made', 'together', 'when', 'eight',
'thereafter', 'third', 'am', 'well', 'll', 'detail', "couldn't", 'either',
'won', 'where', 'very', 'been', 'she', 'was', 'this', 'front', 'therefore',
'sixty', 'whence', 'beyond', 'were', 'several', 'amongst', 'o', 'three',
'throughout', 're', 'into', 'he', 'shan', 'mustn', 'needn', 'own', 'do',
'anyone', 'first', 'almost', 'due', 'system', 'than', 'con', 'fifteen',
'eleven', 'enough', "needn't", 'mightn', 'most', 'more', 'are', 'everywhere',
'thin', 'that', 'yourselves', 'them', 'fill', 'nothing', 'having', 'at',
"didn't", 'may', 'on', 'top', 'became', 'you', 'any', 'take', 'their', 'during',
'only', 'neither', 'whatever', 'us', 'none', 'have', 'both', 'hereupon', 'five',
'cannot', 'mill', 'although', 'co', 'from', 'somehow', 'moreover', 'onto', 'm',
'nevertheless', 'some', 'please', 'too', 'and', 'except', 'even', 'go',
'himself', 'yourself', 'hers', 'bottom', 'un', 'whether', 'another', 'around',
"haven't", 'nor', 'such', "hasn't", 'beside', 'whose', 'then', 'two', 'being',
'aren', 'had', 'full', 'whom', 'ten', 'hereafter', 'could', 'there', 'else',
'rather', 'him', 'itself', 'her', 'your', 'thereupon', 'my', 'mine', 'move',
'but', 'ever', 'describe', 'show', 'afterwards', 'noone', 'six', 'thereby',
'we', 'be', "isn't", 'name', 'would', 'a', "it's", 'anywhere', 'anyhow', 'for',
'towards', "wasn't", 'so', 'off', 'yours', 'four', 'without', 'becoming',
'whereafter', "that'll", 'across', 'everyone', 'fifty', 'myself', 'yet',
'until', 'part', 'least', 'nobody', 'must', 'between', "should've", 'above',
"hadn't", "wouldn't", 'back', 'does', 'forty']}
Run time: 1804.328807592392 seconds
```

[ ]:
```python
#test fit prior => does not improve
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_library)],
    'vect__ngram_range':[(1,1)],
    "clf__alpha" : [0.01],
    "clf__fit_prior" : [True,False],
    'selecter__k':[5000]
}

vectorizer = TfidfVectorizer()
normalizer = Normalizer()
selecter = SelectKBest(chi2)
```

```
pipe = Pipeline([("vect", vectorizer),("normalizer", normalizer),("selecter",␣
 ↪selecter),("clf", MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
The best accuracy is 92.058.
The winning parameters are {'clf__alpha': 0.01, 'clf__fit_prior': True,
'selecter__k': 5000, 'vect__binary': False, 'vect__ngram_range': (1, 1),
'vect__stop_words': ['against', 'find', "shan't", 'i', 't', 'whence', 'go',
'ten', 'she', 'somewhere', 'others', 'throughout', "don't", 'serious',
'whereafter', 'own', 'whole', 'should', 'eg', 'his', 'toward', 'whether',
'wherever', 'give', 'its', 'noone', 'is', 'were', "needn't", 'though',
'therein', 'afterwards', 'everywhere', "doesn't", 'ourselves', "you'll",
'found', 'isn', 'into', "hadn't", 'once', 'are', 'to', 'as', 'down', 'can',
'three', 'don', "wasn't", 'twenty', 'yourselves', 'please', 'often', 'ie', 'an',
'one', 'forty', 'within', 'didn', 'side', 'mightn', 'while', 'sometime', 'hadn',
'all', 'only', "didn't", 'anyone', 'becoming', 'the', 'bottom', 'from',
'almost', 'still', 'describe', 'about', 'anyway', 'd', 'may', 'six', "that'll",
'everything', 'take', 'back', 'for', "isn't", 'mostly', 'eleven', 'whoever',
'whereas', 'moreover', 'why', 'otherwise', 'thus', "she's", 'whose', 'if',
'therefore', 'yet', 'become', 'even', 'five', 'first', 'in', 'something',
'together', 'inc', 'further', 'fill', 'elsewhere', 'very', 'whom', 'each',
'beside', 'some', 'have', 'con', 'latterly', 'themselves', 'hereupon', "it's",
'third', 'upon', 'seems', 'll', 'along', 'itself', 'indeed', 'seem', 'that',
'across', 'will', 'already', 'seemed', 'least', 'becomes', 'show', 'been',
"aren't", 'couldnt', 'mill', 'it', 'except', 'because', 'nowhere', 'by',
'empty', 'out', 'but', 'after', 'beforehand', 'thereby', 'although', 'full',
"haven't", 'latter', 'four', 'then', 'hence', 'her', 'see', 'could', 'you',
'these', 'none', 'thereupon', 'hereafter', 'per', 'shouldn', 'how', 'thence',
'was', 'those', 'nothing', 'perhaps', 'mustn', 'hers', 'doesn', 'there', 'nine',
'ma', 'whither', 'this', 'anyhow', 'interest', 'be', 'o', 'too', 'front',
'less', 'due', 'call', 'rather', 'just', 'without', 'name', 'everyone', 'being',
'over', 'when', 'him', "mightn't", 's', 'amongst', 'amoungst', 'more', 'does',
'formerly', 'de', 'now', 'made', 'hundred', 'below', "you'd", 'through',
```

```
'anywhere', 'sincere', 'of', 'meanwhile', 'thin', 'behind', 'whenever', 'wasn',
'nor', 'until', 'among', 'so', 'yours', 'whereby', 'such', "shouldn't",
'sometimes', 'what', 'thru', 'much', 'same', 'must', 'again', 'a', 'am', 'off',
'never', 'ain', 'they', 'herself', 'etc', 'wouldn', 'thereafter', 'few',
"you've", 'amount', 'namely', 'get', 'yourself', 'besides', 'my', 'than',
'alone', 'couldn', 'might', 'their', 'two', 'between', "won't", 'most', 'them',
"weren't", 'herein', 'and', 'part', 'nevertheless', 'where', 'co', 'another',
'cant', 'bill', 'other', 'fire', 'several', 'did', 'no', 'up', 'cry',
"should've", 'do', 'beyond', 'needn', 'neither', 'next', 'always', 'mine',
'put', 'wherein', 'hasn', "couldn't", 'onto', "you're", "hasn't", 'during',
'however', 'aren', 'thick', 'also', 'm', 'move', 'before', 'doing', 'un',
'which', 'with', 'keep', 'whereupon', 'anything', 'cannot', 'system', 'us',
'done', 'both', "wouldn't", 'here', 'ever', 'enough', 've', "mustn't",
'towards', 'having', 'either', 'hasnt', 'who', 'under', 'fifty', 'haven',
'fifteen', 'eight', 'me', 'former', 'he', 'hereby', 'became', 'or', 'top',
'any', 're', 'has', 'we', 'seeming', 'someone', 'ours', 'else', 'myself',
'above', 'since', 'had', 'our', 'your', 'not', 'would', 'many', 'around',
'detail', 'on', 'sixty', 'somehow', 'at', 'nobody', 'via', 'y', 'shan',
'twelve', 'theirs', 'last', 'ltd', 'every', 'himself', 'whatever', 'won',
'well', 'weren']}
Run time: 3.5223331451416016 seconds
```

[ ]:
```python
#final test before selecting 93.17
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_library)],
    'vect__preprocessor': [preprocess_text,remove_punctuation,None],
    'vect__ngram_range':[(1,2)],
    "clf__alpha" : [0.5]
}

vectorizer = CountVectorizer()



pipe = Pipeline([("vect", vectorizer),("clf", MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
```

```
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
y_pred = grid.predict(test_x)
create_test_csv(y_pred,"MultinomialNB_93.csv")
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
The best accuracy is 93.176.
The winning parameters are {'clf__alpha': 0.5, 'vect__binary': False,
'vect__ngram_range': (1, 2), 'vect__preprocessor': None, 'vect__stop_words':
['against', 'find', "shan't", 'i', 't', 'whence', 'go', 'ten', 'she',
'somewhere', 'others', 'throughout', "don't", 'serious', 'whereafter', 'own',
'whole', 'should', 'eg', 'his', 'toward', 'whether', 'wherever', 'give', 'its',
'noone', 'is', 'were', "needn't", 'though', 'therein', 'afterwards',
'everywhere', "doesn't", 'ourselves', "you'll", 'found', 'isn', 'into',
"hadn't", 'once', 'are', 'to', 'as', 'down', 'can', 'three', 'don', "wasn't",
'twenty', 'yourselves', 'please', 'often', 'ie', 'an', 'one', 'forty', 'within',
'didn', 'side', 'mightn', 'while', 'sometime', 'hadn', 'all', 'only', "didn't",
'anyone', 'becoming', 'the', 'bottom', 'from', 'almost', 'still', 'describe',
'about', 'anyway', 'd', 'may', 'six', "that'll", 'everything', 'take', 'back',
'for', "isn't", 'mostly', 'eleven', 'whoever', 'whereas', 'moreover', 'why',
'otherwise', 'thus', "she's", 'whose', 'if', 'therefore', 'yet', 'become',
'even', 'five', 'first', 'in', 'something', 'together', 'inc', 'further',
'fill', 'elsewhere', 'very', 'whom', 'each', 'beside', 'some', 'have', 'con',
'latterly', 'themselves', 'hereupon', "it's", 'third', 'upon', 'seems', 'll',
'along', 'itself', 'indeed', 'seem', 'that', 'across', 'will', 'already',
'seemed', 'least', 'becomes', 'show', 'been', "aren't", 'couldnt', 'mill', 'it',
'except', 'because', 'nowhere', 'by', 'empty', 'out', 'but', 'after',
'beforehand', 'thereby', 'although', 'full', "haven't", 'latter', 'four',
'then', 'hence', 'her', 'see', 'could', 'you', 'these', 'none', 'thereupon',
'hereafter', 'per', 'shouldn', 'how', 'thence', 'was', 'those', 'nothing',
'perhaps', 'mustn', 'hers', 'doesn', 'there', 'nine', 'ma', 'whither', 'this',
'anyhow', 'interest', 'be', 'o', 'too', 'front', 'less', 'due', 'call',
'rather', 'just', 'without', 'name', 'everyone', 'being', 'over', 'when', 'him',
"mightn't", 's', 'amongst', 'amoungst', 'more', 'does', 'formerly', 'de', 'now',
'made', 'hundred', 'below', "you'd", 'through', 'anywhere', 'sincere', 'of',
'meanwhile', 'thin', 'behind', 'whenever', 'wasn', 'nor', 'until', 'among',
'so', 'yours', 'whereby', 'such', "shouldn't", 'sometimes', 'what', 'thru',
'much', 'same', 'must', 'again', 'a', 'am', 'off', 'never', 'ain', 'they',
'herself', 'etc', 'wouldn', 'thereafter', 'few', "you've", 'amount', 'namely',
'get', 'yourself', 'besides', 'my', 'than', 'alone', 'couldn', 'might', 'their',
'two', 'between', "won't", 'most', 'them', "weren't", 'herein', 'and', 'part',
'nevertheless', 'where', 'co', 'another', 'cant', 'bill', 'other', 'fire',
'several', 'did', 'no', 'up', 'cry', "should've", 'do', 'beyond', 'needn',
'neither', 'next', 'always', 'mine', 'put', 'wherein', 'hasn', "couldn't",
```

```
'onto', "you're", "hasn't", 'during', 'however', 'aren', 'thick', 'also', 'm',
'move', 'before', 'doing', 'un', 'which', 'with', 'keep', 'whereupon',
'anything', 'cannot', 'system', 'us', 'done', 'both', "wouldn't", 'here',
'ever', 'enough', 've', "mustn't", 'towards', 'having', 'either', 'hasnt',
'who', 'under', 'fifty', 'haven', 'fifteen', 'eight', 'me', 'former', 'he',
'hereby', 'became', 'or', 'top', 'any', 're', 'has', 'we', 'seeming', 'someone',
'ours', 'else', 'myself', 'above', 'since', 'had', 'our', 'your', 'not',
'would', 'many', 'around', 'detail', 'on', 'sixty', 'somehow', 'at', 'nobody',
'via', 'y', 'shan', 'twelve', 'theirs', 'last', 'ltd', 'every', 'himself',
'whatever', 'won', 'well', 'weren']}
Run time: 4.155819654464722 seconds
File saved.
```

[ ]:
```
stop_words_custom = [
# All pronouns and associated words
"i","i'll","i'd","i'm","i've","ive","me","myself","you","you'll","you'd","you're","you've","yo
"he'd",
"he's",
"him",
"she",
"she'll",
"she'd",
"she's",
"her",
"it",
"it'll",
"it'd",
"it's",
"itself",
"oneself",
"we",
"we'll",
"we'd",
"we're",
"we've",
"us",
"ourselves",
"they",
"they'll",
"they'd",
"they're",
"they've",
"them",
"themselves",
"everyone",
"everyone's",
"everybody",
```

```
"everybody's",
"someone",
"someone's",
"somebody",
"somebody's",
"nobody",
"nobody's",
"anyone",
"anyone's",
"everything",
"everything's",
"something",
"something's",
"nothing",
"nothing's",
"anything",
"anything's",
# All determiners and associated words
"a",
"an",
"the",
"this",
"that",
"that's",
"these",
"those",
"my",
#"mine",   #Omitted since mine can refer to something else
"your",
"yours",
"his",
"hers",
"its",
"our",
"ours",
"own",
"their",
"theirs",
"few",
"much",
"many",
"lot",
"lots",
"some",
"any",
"enough",
"all",
```

```
"both",
"half",
"either",
"neither",
"each",
"every",
"certain",
"other",
"another",
"such",
"several",
"multiple",
# "what",#Dealt with later on
"rather",
"quite",
# All prepositions
"aboard",
"about",
"above",
"across",
"after",
"against",
"along",
"amid",
"amidst",
"among",
"amongst",
"anti",
"around",
"as",
"at",
"away",
"before",
"behind",
"below",
"beneath",
"beside",
"besides",
"between",
"beyond",
"but",
"by",
"concerning",
"considering",
"despite",
"down",
"during",
```

```
"except",
"excepting",
"excluding",
"far",
"following",
"for",
"from",
"here",
"here's",
"in",
"inside",
"into",
"left",
"like",
"minus",
"near",
"of",
"off",
"on",
"onto",
"opposite",
"out",
"outside",
"over",
"past",
"per",
"plus",
"regarding",
"right",
#"round",   #Omitted
#"save",#Omitted
"since",
"than",
"there",
"there's",
"through",
"to",
"toward",
"towards",
"under",
"underneath",
"unlike",
"until",
"up",
"upon",
"versus",
"via",
```

```
"with",
"within",
"without",
# Irrelevant verbs
"may",
"might",
"will",
"won't",
"would",
"wouldn't",
"can",
"can't",
"cannot",
"could",
"couldn't",
"should",
"shouldn't",
"must",
"must've",
"be",
"being",
"been",
"am",
"are",
"aren't",
"ain't",
"is",
"isn't",
"was",
"wasn't",
"were",
"weren't",
"do",
"doing",
"don't",
"does",
"doesn't",
"did",
"didn't",
"done",
"have",
"haven't",
"having",
"has",
"hasn't",
"had",
"hadn't",
```

```
"get",
"getting",
"gets",
"got",
"gotten",
"go",
"going",
"gonna",
"goes",
"went",
"gone",
"make",
"making",
"makes",
"made",
"take",
"taking",
"takes",
"took",
"taken",
"need",
"needing",
"needs",
"needed",
"use",
"using",
"uses",
"used",
"want",
"wanna",
"wanting",
"wants",
"let",
"lets",
"letting",
"let's",
"suppose",
"supposing",
"supposes",
"supposed",
"seem",
"seeming",
"seems",
"seemed",
"say",
"saying",
"says",
```

```
"said",
"know",
"knowing",
"knows",
"knew",
"known",
"look",
"looking",
"looked",
"think",
"thinking",
"thinks",
"thought",
"feel",
"feels",
"felt",
"based",
"put",
"puts",
#"wanted"   #Omitted since the advective is relevant
# Question words and associated words
"who",
"who's",
"who've",
"who'd",
"whoever",
"whoever's",
"whom",
"whomever",
"whomever's",
"whose",
"whosever",
"whosever's",
"when",
"whenever",
"which",
"whichever",
"where",
"where's",
"where'd",
"wherever",
"why",
"why's",
"why'd",
"whyever",
"what",
"what's",
```

```
"whatever",
"whence",
"how",
"how's",
"how'd",
"however",
"whether",
"whatsoever",
# Connector words and irrelevant adverbs
"and",
"or",
"not",
"because",
"also",
"always",
"never",
"only",
"really",
"very",
"greatly",
"extremely",
"somewhat",
"no",
"nope",
"nah",
"yes",
"yep",
"yeh",
"yeah",
"maybe",
"perhaps",
"more",
"most",
"less",
"least",
"good",
"great",
"well",
"better",
"best",
"bad",
"worse",
"worst",
"too",
"thru",
"though",
"although",
```

```
"yet",
"already",
"then",
"even",
"now",
"sometimes",
"still",
"together",
"altogether",
"entirely",
"fully",
"entire",
"whole",
"completely",
"utterly",
"seemingly",
"apparently",
"clearly",
"obviously",
"actually",
"actual",
"usually",
"usual",
"literally",
"honestly",
"absolutely",
"definitely",
"generally",
"totally",
"finally",
"basically",
"essentially",
"fundamentally",
"automatically",
"immediately",
"necessarily",
"primarily",
"normally",
"perfectly",
"constantly",
"particularly",
"eventually",
"hopefully",
"mainly",
"typically",
"specifically",
"differently",
```

```
"appropriately",
"plenty",
"certainly",
"unfortunately",
"ultimately",
"unlikely",
"likely",
"potentially",
"fortunately",
"personally",
"directly",
"indirectly",
"nearly",
"closely",
"slightly",
"probably",
"possibly",
"especially",
"frequently",
"often",
"oftentimes",
"seldom",
"rarely",
"sure",
"while",
"whilst",
"able",
"unable",
"else",
"ever",
"once",
"twice",
"thrice",
"almost",
"again",
"instead",
"next",
"previous",
"unless",
"somehow",
"anyhow",
"anywhere",
"somewhere",
"everywhere",
"nowhere",
"further",
"anymore",
```

```
"later",
"ago",
"ahead",
"just",
"same",
"different",
"big",
"small",
"little",
"tiny",
"large",
"huge",
"pretty",
"mostly",
"anyway",
"anyways",
"otherwise",
"regardless",
"throughout",
"additionally",
"moreover",
"furthermore",
"meanwhile",
"afterwards",
# Irrelevant nouns
"thing",
"thing's",
"things",
"stuff",
"other's",
"others",
"another's",
"total",
"",
"false",
"none",
"way",
"kind",
# Lettered numbers and order
"zero",
"zeros",
"zeroes",
"one",
"ones",
"two",
"three",
"four",
```

```
"five",
"six",
"seven",
"eight",
"nine",
"ten",
"twenty",
"thirty",
"forty",
"fifty",
"sixty",
"seventy",
"eighty",
"ninety",
"hundred",
"hundreds",
"thousand",
"thousands",
"million",
"millions",
"first",
"last",
"second",
"third",
"fourth",
"fifth",
"sixth",
"seventh",
"eigth",
"ninth",
"tenth",
"firstly",
"secondly",
"thirdly",
"lastly",
# Greetings and slang
"hello",
"hi",
"hey",
"sup",
"yo",
"greetings",
"please",
"okay",
"ok",
"y'all",
"lol",
```

```
"rofl",
"thank",
"thanks",
"alright",
"kinda",
"dont",
"sorry",
"idk",
"tldr",
"tl",
"dr",  #This means that dr (doctor) is a bad feature because of tl;dr
"tbh",
"dude",
"tho",
"aka",
"plz",
"pls",
"bit",
"don",
# Miscellaneous
"www",
"https",
"http",
"com",
"etc"
"html",
"reddit",
"subreddit",
"subreddits",
"comments",
"reply",
"replies",
"thread",
"threads",
"post",
"posts",
"website",
"websites",
"web site",
"web sites"]
print('length custom:',len(stop_words_custom))
```

```
length custom: 589
```

[ ]:
```
#test custom dictionary => 94.01
#selected =>4
t_start = time.time()
```

```
pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_custom)],
    'vect__preprocessor': [preprocess_text,remove_punctuation,None],
    'vect__ngram_range':[(1,1)],
    "clf__alpha" : [0.5]
}

vectorizer = CountVectorizer()



pipe = Pipeline([("vect", vectorizer),("clf", MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
y_pred = grid.predict(test_x)
create_test_csv(y_pred,"MultinomialNB_without.csv")
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits

/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 94.011.
The winning parameters are {'clf__alpha': 0.5, 'vect__binary': False,
'vect__ngram_range': (1, 1), 'vect__preprocessor': None, 'vect__stop_words':
['i', "i'll", "i'd", "i'm", "i've", 'ive', 'me', 'myself', 'you', "you'll",
"you'd", "you're", "you've", 'yourself', 'he', "he'll", "he'd", "he's", 'him',
'she', "she'll", "she'd", "she's", 'her', 'it', "it'll", "it'd", "it's",
'itself', 'oneself', 'we', "we'll", "we'd", "we're", "we've", 'us', 'ourselves',
'they', "they'll", "they'd", "they're", "they've", 'them', 'themselves',
```

'everyone', "everyone's", 'everybody', "everybody's", 'someone', "someone's",
'somebody', "somebody's", 'nobody', "nobody's", 'anyone', "anyone's",
'everything', "everything's", 'something', "something's", 'nothing',
"nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this', 'that',
"that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its', 'our',
'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots', 'some',
'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each', 'every',
'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather', 'quite',
'aboard', 'about', 'above', 'across', 'after', 'against', 'along', 'amid',
'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away', 'before',
'behind', 'below', 'beneath', 'beside', 'besides', 'between', 'beyond', 'but',
'by', 'concerning', 'considering', 'despite', 'down', 'during', 'except',
'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here', "here's",
'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off', 'on',
'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',

```
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites']}
Run time: 5.304453372955322 seconds
File saved.
```

[ ]:
```
#test custom dictionary => 94.01
#selected =>4
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_custom)],
    #'vect__preprocessor': [preprocess_text,remove_punctuation,None],
    'vect__preprocessor': [remove_punctuation],
    'vect__ngram_range':[(1,1)],
    "clf__alpha" : [0.5]
}

vectorizer = CountVectorizer()



pipe = Pipeline([("vect", vectorizer),("clf", MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)
```

```
grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['aint', 'anothers', 'anyones',
'anythings', 'arent', 'cant', 'couldnt', 'didnt', 'doesnt', 'everybodys',
'everyones', 'everythings', 'hadnt', 'hasnt', 'havent', 'hed', 'hell', 'heres',
'hes', 'howd', 'hows', 'id', 'ill', 'im', 'isnt', 'itd', 'itll', 'mustve',
'nobodys', 'nothings', 'shed', 'shell', 'shes', 'shouldnt', 'site', 'sites',
'somebodys', 'someones', 'somethings', 'thats', 'theres', 'theyd', 'theyll',
'theyre', 'theyve', 'wasnt', 'web', 'wed', 'werent', 'weve', 'whats', 'whered',
'wheres', 'whod', 'whoevers', 'whomevers', 'whos', 'whosevers', 'whove', 'whyd',
'whys', 'wont', 'wouldnt', 'yall', 'youd', 'youll', 'youre', 'youve'] not in
stop_words.
  warnings.warn(
```

```
The best accuracy is 89.558.
The winning parameters are {'clf__alpha': 0.5, 'vect__binary': False,
'vect__ngram_range': (1, 1), 'vect__preprocessor': <function remove_punctuation
at 0x7f6bd75ee790>, 'vect__stop_words': ['i', "i'll", "i'd", "i'm", "i've",
'ive', 'me', 'myself', 'you', "you'll", "you'd", "you're", "you've", 'yourself',
'he', "he'll", "he'd", "he's", 'him', 'she', "she'll", "she'd", "she's", 'her',
'it', "it'll", "it'd", "it's", 'itself', 'oneself', 'we', "we'll", "we'd",
"we're", "we've", 'us', 'ourselves', 'they', "they'll", "they'd", "they're",
"they've", 'them', 'themselves', 'everyone', "everyone's", 'everybody',
"everybody's", 'someone', "someone's", 'somebody', "somebody's", 'nobody',
"nobody's", 'anyone', "anyone's", 'everything', "everything's", 'something',
"something's", 'nothing', "nothing's", 'anything', "anything's", 'a', 'an',
'the', 'this', 'that', "that's", 'these', 'those', 'my', 'your', 'yours', 'his',
'hers', 'its', 'our', 'ours', 'own', 'their', 'theirs', 'few', 'much', 'many',
'lot', 'lots', 'some', 'any', 'enough', 'all', 'both', 'half', 'either',
'neither', 'each', 'every', 'certain', 'other', 'another', 'such', 'several',
'multiple', 'rather', 'quite', 'aboard', 'about', 'above', 'across', 'after',
'against', 'along', 'amid', 'amidst', 'among', 'amongst', 'anti', 'around',
'as', 'at', 'away', 'before', 'behind', 'below', 'beneath', 'beside', 'besides',
'between', 'beyond', 'but', 'by', 'concerning', 'considering', 'despite',
'down', 'during', 'except', 'excepting', 'excluding', 'far', 'following', 'for',
```

'from', 'here', "here's", 'in', 'inside', 'into', 'left', 'like', 'minus',
'near', 'of', 'off', 'on', 'onto', 'opposite', 'out', 'outside', 'over', 'past',
'per', 'plus', 'regarding', 'right', 'since', 'than', 'there', "there's",
'through', 'to', 'toward', 'towards', 'under', 'underneath', 'unlike', 'until',
'up', 'upon', 'versus', 'via', 'with', 'within', 'without', 'may', 'might',
'will', "won't", 'would', "wouldn't", 'can', "can't", 'cannot', 'could',
"couldn't", 'should', "shouldn't", 'must', "must've", 'be', 'being', 'been',
'am', 'are', "aren't", "ain't", 'is', "isn't", 'was', "wasn't", 'were',
"weren't", 'do', 'doing', "don't", 'does', "doesn't", 'did', "didn't", 'done',
'have', "haven't", 'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting',
'gets', 'got', 'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make',
'making', 'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need',
'needing', 'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna',
'wanting', 'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing',
'supposes', 'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying',
'says', 'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',

```
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites']}
Run time: 2.1990275382995605 seconds
```

[ ]:
```
y_pred_new = grid.predict(test_x)
create_test_csv(y_pred_new,"multi_pipeline.csv")
```

```
File saved.
```

[ ]:
```
#test selector => 94.011.
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_custom)],
    'vect__preprocessor': [None],
    'vect__ngram_range':[(1,1)],
    "selecter__k":[5000,3000],
    "clf__alpha" : [0.5,0.1],
   #  "normalizer__norm": ['l2','l1']
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
#normalizer = Normalizer()




#pipe = Pipeline([("vect", vectorizer),("selecter",␣
 ↪selecter),("normalizer",normalizer),("clf", MultinomialNB())])
pipe = Pipeline([("vect", vectorizer),("selecter", selecter),("clf",␣
 ↪MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()
```

```
elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")


y_pred = grid.predict(test_x)
create_test_csv(y_pred,"MultinomialNB_S_03032023_01.csv")
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(
```

```
The best accuracy is 94.011.
The winning parameters are {'clf__alpha': 0.5, 'selecter__k': 5000,
'vect__binary': False, 'vect__ngram_range': (1, 1), 'vect__preprocessor': None,
'vect__stop_words': ['i', "i'll", "i'd", "i'm", "i've", 'ive', 'me', 'myself',
'you', "you'll", "you'd", "you're", "you've", 'yourself', 'he', "he'll", "he'd",
"he's", 'him', 'she', "she'll", "she'd", "she's", 'her', 'it', "it'll", "it'd",
"it's", 'itself', 'oneself', 'we', "we'll", "we'd", "we're", "we've", 'us',
'ourselves', 'they', "they'll", "they'd", "they're", "they've", 'them',
'themselves', 'everyone', "everyone's", 'everybody', "everybody's", 'someone',
"someone's", 'somebody', "somebody's", 'nobody', "nobody's", 'anyone',
"anyone's", 'everything', "everything's", 'something', "something's", 'nothing',
"nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this', 'that',
"that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its', 'our',
'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots', 'some',
'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each', 'every',
'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather', 'quite',
'aboard', 'about', 'above', 'across', 'after', 'against', 'along', 'amid',
'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away', 'before',
'behind', 'below', 'beneath', 'beside', 'besides', 'between', 'beyond', 'but',
'by', 'concerning', 'considering', 'despite', 'down', 'during', 'except',
'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here', "here's",
'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off', 'on',
'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
```

"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',

```
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites']}
Run time: 4.01872992515564 seconds
File saved.
```

[ ]:
```
#test the final after preprocessing
t_start = time.time()

pipe_params = {
    "vect__binary": [False],
    "vect__stop_words": [list(stop_words_custom)],
    'vect__preprocessor': [],
    'vect__ngram_range':[(1,1)],
    "selecter__k":[5000,3000],
    "clf__alpha" : [0.5,0.1],
   #  "normalizer__norm": ['l2','l1']
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
#normalizer = Normalizer()




#pipe = Pipeline([("vect", vectorizer),("selecter",␣
 ↪selecter),("normalizer",normalizer),("clf", MultinomialNB())])
pipe = Pipeline([("vect", vectorizer),("selecter", selecter),("clf",␣
 ↪MultinomialNB())])


grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")


y_pred = grid.predict(test_x)
```

```
create_test_csv(y_pred,"MultinomialNB_S_03032023_01.csv")
```

[ ]:
```
#now that the model is finalized , build the final model

#####################################################################do not use this

from sklearn.model_selection import cross_val_score


final_vectorize = CountVectorizer(stop_words = stop_words_custom,␣
 ↪ngram_range=(1,1), binary=False)
vec_x_train = np.asarray(final_vectorize.fit_transform(train_x).todense())
vec_x_test = np.asarray(final_vectorize.transform(test_x).todense())

#skLearnFeatureSelector = SelectKBest(chi2, k=5000)

#selected_x_train = skLearnFeatureSelector.fit_transform(vec_x_train, train_y)
#selected_x_test = skLearnFeatureSelector.transform(vec_x_test)



model = MultinomialNB(alpha=0.5)
model.fit(vec_x_train, train_y)

# Step 4: Evaluate the model using cross-validation
cv_scores = cross_val_score(model, vec_x_train, train_y, cv=5)
mean_cv_accuracy = np.mean(cv_scores)

print(f"The 5-fold cross-validation accuracy is: {mean_cv_accuracy:.5f}")


y_pred = model.predict(vec_x_test)
create_test_csv(y_pred,"final_MultinomialNB.csv")
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The 5-fold cross-validation accuracy is: 0.92755
File saved.
```

[ ]:
```
########################################################## final
```

SVM

March 12, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from google.colab import drive
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import random
import time
import re
import string
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2,␣
 ↪f_classif,mutual_info_classif,f_regression
from sklearn.preprocessing import Normalizer
from sklearn import model_selection
from sklearn import svm
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
nltk.download('wordnet')
nltk.download('stopwords')
from sklearn.svm import SVC
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[2]:
```
#import the data
drive.mount('/content/gdrive/', force_remount=True)

train_data_initial = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/train.
 ↪csv')
test_data = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/test.csv')

print('shape train:',train_data_initial.shape)
print('shape test:',test_data.shape)
```

```
Mounted at /content/gdrive/
shape train: (718, 2)
shape test: (279, 2)
```

[3]:
```
def shuffle_data(df):
    random.seed(0)  # Use a fixed seed for the random number generator
    df = df.sample(frac=1, random_state=0).reset_index(drop=True)
    return df
```

[4]:
```
#function for creating the test csv file to upload to kaggle
def create_test_csv(data, outfile_name):
  rawdata= {'subreddit':data}
  csv = pd.DataFrame(rawdata, columns = ['subreddit'])
  csv.to_csv(outfile_name,index=True, header=True)
  print ("File saved.")
```

[9]:
```
#shuffle the data and split the features from the label
train_data = shuffle_data(train_data_initial)

train_x = train_data["body"]
train_y = train_data["subreddit"]
test_x = test_data["body"]
```

[6]:
```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    return text
```

[7]:
```
#create a dictionary of stop words
stop_words_nltk = set(stopwords.words('english'))
stop_words_sklearn = text.ENGLISH_STOP_WORDS
stop_words_library = stop_words_sklearn.union(stop_words_nltk)
```

[ ]:
```
############################################################
```

[11]:
```
#initial training without removing parameters
t_start = time.time()

pipe_params = {
    'classify__C': [0.1, 1, 10],
    'classify__kernel': ['linear', 'rbf']
}

vectorizer = CountVectorizer()
model = SVC()

pipe = Pipeline(
    [("vect", vectorizer),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
The best accuracy is 90.251.
The winning parameters are {'classify__C': 10, 'classify__kernel': 'rbf'}
Run time: 15.285731792449951 seconds
```

[13]:
```
#testing stop words
t_start = time.time()

pipe_params = {
    "vect__binary": [False,True],
    "vect__stop_words":␣
 ↪[list(stop_words_nltk),list(stop_words_sklearn),list(stop_words_library)],
    "selecter__k":[5000,6000,3000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)


pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter),("classify", SVC())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
The best accuracy is 92.198.
The winning parameters are {'classify__alpha': 0.1, 'selecter__k': 5000,
'vect__binary': False, 'vect__stop_words': ['when', 'few', 'very', 'between',
'nine', 'd', 'elsewhere', 'ourselves', 'wherein', 'several', 'still', 'even',
'seeming', 'an', 'becoming', 'below', 'give', 'nobody', 'behind', 'thru',
'mustn', 'ma', 'about', 'if', 'must', 'toward', 'what', 'on', 'through', 'ever',
'anyhow', 'there', 'fill', 'empty', 'by', 'these', 'co', 'full', 'therefore',
"didn't", 'won', 'you', 'another', 'within', 'seemed', 'sometimes', 'doesn',
'meanwhile', 'becomes', 'thence', 'fifteen', 'take', 'to', 'will', 'hadn',
'found', 'have', 'four', 'them', 'whereby', 'were', 'theirs', 'be', "wasn't",
```

```
'six', 'nevertheless', 'formerly', 'are', 'although', 'cry', 'sometime', 'eg',
'further', 'perhaps', 'again', 'under', 'this', 'alone', 'us', 'might', 'see',
'do', 'both', 'against', 'con', 'having', 'since', 'around', 'needn', 'himself',
'system', 'among', 'eleven', 'for', 'former', 'it', 'onto', 'interest',
'anyway', 'hereby', 'out', 'after', 'itself', "don't", "isn't", 'before',
'besides', "should've", 'nothing', 'can', "you'll", 'or', 'get', 'anyone',
'move', 'whence', 'mine', 'a', 'nor', 'other', 'per', 'back', 'last',
'everywhere', 'm', 'part', 'own', 'name', 'should', 'y', 'was', 'didn',
'whereupon', 'mightn', 'over', 'haven', 'bottom', "hadn't", 'thereafter',
'anything', 'inc', 'above', 'de', 'noone', 'don', "it's", 'one', 'from',
'someone', 'during', 'therein', 'everyone', 'well', 'his', 'i', 'being',
'without', 'while', 'o', 'done', 'whatever', 'yet', "shouldn't", 'hence', 'go',
'hasn', 'afterwards', 'seems', 'as', "that'll", 'may', 'though', 'hereafter',
'however', 'made', 'seem', 'him', 'amoungst', 'somehow', 'mostly', 'whither',
'none', 'then', 'could', 'also', 'how', 't', 'off', 'others', 'ie', 'latter',
'serious', 'describe', 'everything', 'across', 'll', 'yourself', 'front',
'same', 'yours', 'next', 'no', 'else', 'via', 'thin', "wouldn't", 'side', 'up',
'every', 'two', 'mill', 'something', 'already', 'together', 'many', 'thus',
'but', 'that', 'rather', 'neither', 'nowhere', 'your', 'its', 'except', 'ten',
'keep', 'show', 'yourselves', 'my', "couldn't", 'where', 'much', 'he', 'herein',
'down', 'wherever', 'with', 'due', 'namely', 'please', 'always', 'did',
"you've", "shan't", 'into', 'cant', 'less', 'five', 'had', 'twelve', 'and',
'along', 'almost', "haven't", 'most', "aren't", 'third', 'some', 'hundred',
'they', 'such', 'been', 're', 'indeed', 'often', 'would', "mightn't", 'just',
'me', 'call', 'weren', 'now', 'of', 'throughout', 'thick', 'whenever', 'until',
'cannot', 'least', 'thereupon', 'beside', 'hers', "doesn't", 'beyond',
'thereby', 'towards', 'couldn', 'top', 'once', 'whole', 'three', 'couldnt',
'ours', 'has', 'more', 'forty', 'whereafter', 'amongst', 'beforehand', 's',
'became', 'fifty', 'wasn', "you'd", 'am', 'twenty', "needn't", 'each', 'does',
'in', 'otherwise', 'ain', 'bill', 'become', 'than', 'detail', 'at', 'put',
'themselves', 'because', 'shan', 'latterly', 'sixty', 'our', 'we', 'eight',
'not', 'amount', 'too', 'fire', 'whereas', 'who', 'doing', 'isn', 'whom', 'any',
'whether', "won't", 'un', 'etc', 'so', 'her', 'shouldn', "weren't", 'myself',
'upon', 'somewhere', 'never', 'which', 'aren', "you're", 'why', 'ltd',
"mustn't", 'hereupon', 'herself', 've', 'whose', 'is', "hasn't", 'enough',
'all', 'only', 'those', 'whoever', 'wouldn', 'anywhere', 'hasnt', 'the',
'their', 'sincere', "she's", 'here', 'either', 'first', 'find', 'moreover',
'she']}
Run time: 74.24770450592041 seconds
```

[ ]:
```
#testing normalizer
t_start = time.time()

pipe_params = {
    "vect__binary": [False,True],
    "vect__stop_words": [list(stop_words_library)],
    "selecter__k":[5000,3000],
```

```
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5],
    "normalizer__norm": ['l2','l1','max']
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
normalizer = Normalizer()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter",␣
 ↪selecter),("normalizer",normalizer),("classify", SVC())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
The best accuracy is 90.53.
The winning parameters are {'classify__alpha': 0.02, 'normalizer__norm': 'l1',
'selecter__k': 5000, 'vect__binary': False, 'vect__stop_words':
frozenset({'only', 'whereby', 'thereby', 'within', 'that', 'top', 'bill',
'here', 'ain', 'anyway', 'himself', 'full', 'there', 'nine', 'well', 'couldn',
'would', 'they', "hadn't", 'along', 'whether', 'more', 'around', 'an', 'hasnt',
'she', 'never', 'be', 'already', 'de', 'else', 'whose', 'anyone', 'wasn',
'without', 'whole', 'thru', 'even', "it's", "doesn't", 'none', 'made', 'to',
'not', 'still', 'sometimes', 'my', 'yours', 'from', 'keep', 'who', "you'd",
'further', 'his', 'might', 'whoever', 'through', 'formerly', 'describe', 'ltd',
'whereafter', 'whenever', 'being', 'us', 'upon', 'ourselves', 'show', 'against',
'we', 'cannot', 'anyhow', 'doing', 'get', 'has', 'with', 'by', 'just',
'seeming', 'whatever', 'although', 'most', 'when', 'thereafter', 'below', 'and',
'yourself', 'how', 'down', 'back', 'five', 'four', "you'll", 'until', 'what',
'while', 'as', 'fill', 'those', "aren't", 'but', 'call', 'could', 'their',
'then', 'noone', 'six', 'which', 'ma', 'throughout', 'anything', 'part',
'itself', 'again', 'twelve', "won't", 'last', 'whereas', 'up', 'perhaps',
'ours', 'all', "needn't", 'been', 'eight', 'etc', 'via', 'amoungst', 'o',
'either', 'least', 'three', "couldn't", 'didn', 'beforehand', 'latter',
'thence', 'amongst', 'hereby', 'whither', 'became', 'couldnt', "wouldn't",
```

```
'onto', 'seem', 'm', 'almost', 'fifty', 'nowhere', 'anywhere', 'therein',
"didn't", 'under', 'was', 'others', "haven't", 'thin', 'behind', 'too', 'done',
'after', 'should', 'third', 'per', 'across', "you're", 'becoming', 'its',
'these', 'them', 've', 'enough', 'if', 'seems', 'beyond', 'fire', 'front',
'somehow', 'however', 'everything', 'a', 'empty', 'y', 'together', 'shouldn',
'for', 'hereupon', 'hadn', 'same', 'hence', 'indeed', 'over', 'no', 'doesn',
'have', 'forty', 'ten', 'amount', 'having', 'hasn', 'any', 'off', 'such',
'first', 'themselves', "don't", 'bottom', 'rather', "mustn't", 'do', 'are',
'can', 'besides', 'somewhere', 'fifteen', "weren't", 'since', 'also', 'mill',
'often', 'nobody', 'due', 'wherever', 'did', 'always', 'thereupon', "hasn't",
'name', 'therefore', 'un', 'go', 'aren', "isn't", 'were', 'out', 'sincere',
"mightn't", 'thick', 'inc', 'become', 'alone', 'several', 'this', 'he', 'among',
'll', 'detail', 'during', 'mostly', 'you', 'won', 'namely', 'our', 'yourselves',
'in', 'why', 'herein', 'wherein', 'serious', 'both', 'the', 'toward',
"shouldn't", 'on', 'another', 'because', 'haven', 'needn', 'please', 'next',
'find', 'your', 'moreover', "should've", 'though', 's', "wasn't", 'nothing',
'less', 'system', 'twenty', 'now', 'about', 'mustn', 'herself', 'hers', 'or',
'every', 'than', 'everywhere', "you've", 'latterly', "she's", 'afterwards',
'weren', 'above', 'side', 'everyone', 'eg', 'elsewhere', 're', 'hereafter',
'where', 'see', 'very', 'yet', 'myself', 'two', 'former', 'cry', 'towards',
'thus', 'i', 'd', 'ie', 'whence', 'con', 'move', 'mightn', 'am', 'don',
'hundred', 'of', 'whereupon', 'other', 'once', 'me', 'her', 'wouldn',
'otherwise', 'found', 'seemed', 'give', 'becomes', 'it', 'at', 'between',
'something', 'so', 'him', 'into', 'neither', 't', 'put', 'except', 'few',
'beside', 'whom', 'meanwhile', 'nevertheless', 'mine', 'isn', 'does', 'before',
'may', 'ever', 'theirs', 'will', 'eleven', "that'll", 'one', "shan't", 'take',
'sixty', 'sometime', 'each', 'had', 'interest', 'own', 'is', 'much', 'shan',
'cant', 'nor', 'co', 'many', 'must', 'some', 'someone'})}
Run time: 26.246994018554688 seconds
```

[14]:
```python
#stem lemmatizer
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

class LemmaTokenizer_Pos:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in␣
 ↪word_tokenize(doc) if t.isalpha()]
```

```
class LemmaTokenizer:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) if t.
 ↪isalpha()]

class LemmaTokenizer_word:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) ]


class StemTokenizer:
     def __init__(self):
       self.wnl =PorterStemmer()
     def __call__(self, doc):
       return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
```

[ ]:
```
#testing lemma
t_start = time.time()

pipe_params = {
    "vect__binary": [False,True],
    "vect__stop_words": [list(stop_words_library)],
    "vect__tokenizer": [LemmaTokenizer_word()],
    "selecter__k":[5000,3000]
    }

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
normalizer = Normalizer()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter",␣
 ↪selecter),("normalizer",normalizer),("classify", SVC())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)
```

```
print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:396:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'ll", "'re", "'s", "'ve",
'make', "n't", 'need', 'sha', 'win', 'wo'] not in stop_words.
  warnings.warn(
```

```
The best accuracy is 89.833.
The winning parameters are {'classify__alpha': 0.1, 'selecter__k': 5000,
'vect__binary': False, 'vect__stop_words': frozenset({'only', 'whereby',
'thereby', 'within', 'that', 'top', 'bill', 'here', 'ain', 'anyway', 'himself',
'full', 'there', 'nine', 'well', 'couldn', 'would', 'they', "hadn't", 'along',
'whether', 'more', 'around', 'an', 'hasnt', 'she', 'never', 'be', 'already',
'de', 'else', 'whose', 'anyone', 'wasn', 'without', 'whole', 'thru', 'even',
"it's", "doesn't", 'none', 'made', 'to', 'not', 'still', 'sometimes', 'my',
'yours', 'from', 'keep', 'who', "you'd", 'further', 'his', 'might', 'whoever',
'through', 'formerly', 'describe', 'ltd', 'whereafter', 'whenever', 'being',
'us', 'upon', 'ourselves', 'show', 'against', 'we', 'cannot', 'anyhow', 'doing',
'get', 'has', 'with', 'by', 'just', 'seeming', 'whatever', 'although', 'most',
'when', 'thereafter', 'below', 'and', 'yourself', 'how', 'down', 'back', 'five',
'four', "you'll", 'until', 'what', 'while', 'as', 'fill', 'those', "aren't",
'but', 'call', 'could', 'their', 'then', 'noone', 'six', 'which', 'ma',
'throughout', 'anything', 'part', 'itself', 'again', 'twelve', "won't", 'last',
'whereas', 'up', 'perhaps', 'ours', 'all', "needn't", 'been', 'eight', 'etc',
'via', 'amoungst', 'o', 'either', 'least', 'three', "couldn't", 'didn',
'beforehand', 'latter', 'thence', 'amongst', 'hereby', 'whither', 'became',
'couldnt', "wouldn't", 'onto', 'seem', 'm', 'almost', 'fifty', 'nowhere',
'anywhere', 'therein', "didn't", 'under', 'was', 'others', "haven't", 'thin',
'behind', 'too', 'done', 'after', 'should', 'third', 'per', 'across', "you're",
'becoming', 'its', 'these', 'them', 've', 'enough', 'if', 'seems', 'beyond',
'fire', 'front', 'somehow', 'however', 'everything', 'a', 'empty', 'y',
'together', 'shouldn', 'for', 'hereupon', 'hadn', 'same', 'hence', 'indeed',
'over', 'no', 'doesn', 'have', 'forty', 'ten', 'amount', 'having', 'hasn',
'any', 'off', 'such', 'first', 'themselves', "don't", 'bottom', 'rather',
"mustn't", 'do', 'are', 'can', 'besides', 'somewhere', 'fifteen', "weren't",
'since', 'also', 'mill', 'often', 'nobody', 'due', 'wherever', 'did', 'always',
'thereupon', "hasn't", 'name', 'therefore', 'un', 'go', 'aren', "isn't", 'were',
'out', 'sincere', "mightn't", 'thick', 'inc', 'become', 'alone', 'several',
'this', 'he', 'among', 'll', 'detail', 'during', 'mostly', 'you', 'won',
'namely', 'our', 'yourselves', 'in', 'why', 'herein', 'wherein', 'serious',
'both', 'the', 'toward', "shouldn't", 'on', 'another', 'because', 'haven',
'needn', 'please', 'next', 'find', 'your', 'moreover', "should've", 'though',
```

```
's', "wasn't", 'nothing', 'less', 'system', 'twenty', 'now', 'about', 'mustn',
'herself', 'hers', 'or', 'every', 'than', 'everywhere', "you've", 'latterly',
"she's", 'afterwards', 'weren', 'above', 'side', 'everyone', 'eg', 'elsewhere',
're', 'hereafter', 'where', 'see', 'very', 'yet', 'myself', 'two', 'former',
'cry', 'towards', 'thus', 'i', 'd', 'ie', 'whence', 'con', 'move', 'mightn',
'am', 'don', 'hundred', 'of', 'whereupon', 'other', 'once', 'me', 'her',
'wouldn', 'otherwise', 'found', 'seemed', 'give', 'becomes', 'it', 'at',
'between', 'something', 'so', 'him', 'into', 'neither', 't', 'put', 'except',
'few', 'beside', 'whom', 'meanwhile', 'nevertheless', 'mine', 'isn', 'does',
'before', 'may', 'ever', 'theirs', 'will', 'eleven', "that'll", 'one', "shan't",
'take', 'sixty', 'sometime', 'each', 'had', 'interest', 'own', 'is', 'much',
'shan', 'cant', 'nor', 'co', 'many', 'must', 'some', 'someone'}),
'vect__tokenizer': <__main__.LemmaTokenizer_word object at 0x7f4b6a36c940>}
Run time: 74.08984041213989 seconds
```

[ ]:
```
# Step 5: Make predictions on test data using the trained model
```

[ ]:
```
############################################################ final
```

[17]:
```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.svm import SVC
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import numpy as np



# define the stop words
stop_words = set(stopwords.words('english'))

# define the pipeline
pipeline = Pipeline([
    ('vectorize', CountVectorizer(stop_words=list(stop_words),␣
 ↪binary=True,lowercase = False,preprocessor=preprocess_text)),
    ('selector', SelectKBest(chi2, k=3000)),
    ('clf', SVC())
])

cross_val_score = np.mean(model_selection.cross_val_score(pipeline, train_x,␣
 ↪train_y, cv=5,n_jobs=-1,verbose=1))
print('cross_val_score->',cross_val_score)

pipeline.fit(train_x, train_y)

test_x_processed = pipeline.named_steps['vectorize'].transform(test_x)
```

```
test_x_selected = pipeline.named_steps['selector'].transform(test_x_processed)



y_pred = pipeline.predict(test_x)

create_test_csv(y_pred,"SVM.csv")
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    4.6s finished

cross_val_score-> 0.8551379176379175
File saved.
```

[ ]:

```
File saved.
```

DesicionTree

March 12, 2023

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from google.colab import drive
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import random
import time
import re
import string
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2,␣
 ↪f_classif,mutual_info_classif,f_regression
from sklearn.preprocessing import Normalizer
from sklearn import model_selection
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

nltk.download('punkt')
```

```
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('stopwords')
from sklearn.svm import SVC
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[ ]:
```
#import the data
drive.mount('/content/gdrive/', force_remount=True)

train_data_initial = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/train.
 ↪csv')
test_data = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/test.csv')

print('shape train:',train_data_initial.shape)
print('shape test:',test_data.shape)
```

```
Mounted at /content/gdrive/
shape train: (718, 2)
shape test: (279, 2)
```

[ ]:
```
def shuffle_data(df):
    random.seed(0)  # Use a fixed seed for the random number generator
    df = df.sample(frac=1, random_state=0).reset_index(drop=True)
    return df
```

[ ]:
```
#function for creating the test csv file to upload to kaggle
def create_test_csv(data, outfile_name):
  rawdata= {'subreddit':data}
  csv = pd.DataFrame(rawdata, columns = ['subreddit'])
  csv.to_csv(outfile_name,index=True, header=True)
```

```
  print ("File saved.")
```

[ ]:
```
#shuffle the data and split the features from the label
train_data = shuffle_data(train_data_initial)


train_x = train_data["body"]
train_y = train_data["subreddit"]
test_x = test_data["body"]
```

[ ]:
```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    return text
```

[ ]:
```
#create a dictionary of stop words
stop_words_nltk = set(stopwords.words('english'))
stop_words_sklearn = text.ENGLISH_STOP_WORDS
stop_words_library = list(stop_words_sklearn.union(stop_words_nltk))
```

[ ]:
```
############################################################
```

[ ]:
```
#initial training of DecisionTree
t_start = time.time()

pipe_params = {
    'clf__criterion': ['gini', 'entropy'],
    'clf__max_depth': [10, 50, 100, None],
    'clf__min_samples_split': [2, 5, 10],
    'clf__min_samples_leaf': [1, 2, 4]
}

vectorizer = CountVectorizer()
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)
```

```
print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
The best accuracy is 86.072.
The winning parameters are {'clf__criterion': 'entropy', 'clf__max_depth': 50,
'clf__min_samples_leaf': 4, 'clf__min_samples_split': 5}
Run time: 45.189074993133545 seconds
```

[ ]:
```
stop_words_custom = [
# All pronouns and associated words
"i","i'll","i'd","i'm","i've","ive","me","myself","you","you'll","you'd","you're","you've","yo
"he'd",
"he's",
"him",
"she",
"she'll",
"she'd",
"she's",
"her",
"it",
"it'll",
"it'd",
"it's",
"itself",
"oneself",
"we",
"we'll",
"we'd",
"we're",
"we've",
"us",
"ourselves",
"they",
"they'll",
"they'd",
"they're",
"they've",
"them",
"themselves",
"everyone",
"everyone's",
"everybody",
"everybody's",
"someone",
```

```
"someone's",
"somebody",
"somebody's",
"nobody",
"nobody's",
"anyone",
"anyone's",
"everything",
"everything's",
"something",
"something's",
"nothing",
"nothing's",
"anything",
"anything's",
# All determiners and associated words
"a",
"an",
"the",
"this",
"that",
"that's",
"these",
"those",
"my",
#"mine",   #Omitted since mine can refer to something else
"your",
"yours",
"his",
"hers",
"its",
"our",
"ours",
"own",
"their",
"theirs",
"few",
"much",
"many",
"lot",
"lots",
"some",
"any",
"enough",
"all",
"both",
"half",
```

```
"either",
"neither",
"each",
"every",
"certain",
"other",
"another",
"such",
"several",
"multiple",
# "what",#Dealt with later on
"rather",
"quite",
# All prepositions
"aboard",
"about",
"above",
"across",
"after",
"against",
"along",
"amid",
"amidst",
"among",
"amongst",
"anti",
"around",
"as",
"at",
"away",
"before",
"behind",
"below",
"beneath",
"beside",
"besides",
"between",
"beyond",
"but",
"by",
"concerning",
"considering",
"despite",
"down",
"during",
"except",
"excepting",
```

```
"excluding",
"far",
"following",
"for",
"from",
"here",
"here's",
"in",
"inside",
"into",
"left",
"like",
"minus",
"near",
"of",
"off",
"on",
"onto",
"opposite",
"out",
"outside",
"over",
"past",
"per",
"plus",
"regarding",
"right",
#"round",   #Omitted
#"save",#Omitted
"since",
"than",
"there",
"there's",
"through",
"to",
"toward",
"towards",
"under",
"underneath",
"unlike",
"until",
"up",
"upon",
"versus",
"via",
"with",
"within",
```

```
"without",
# Irrelevant verbs
"may",
"might",
"will",
"won't",
"would",
"wouldn't",
"can",
"can't",
"cannot",
"could",
"couldn't",
"should",
"shouldn't",
"must",
"must've",
"be",
"being",
"been",
"am",
"are",
"aren't",
"ain't",
"is",
"isn't",
"was",
"wasn't",
"were",
"weren't",
"do",
"doing",
"don't",
"does",
"doesn't",
"did",
"didn't",
"done",
"have",
"haven't",
"having",
"has",
"hasn't",
"had",
"hadn't",
"get",
"getting",
```

```
"gets",
"got",
"gotten",
"go",
"going",
"gonna",
"goes",
"went",
"gone",
"make",
"making",
"makes",
"made",
"take",
"taking",
"takes",
"took",
"taken",
"need",
"needing",
"needs",
"needed",
"use",
"using",
"uses",
"used",
"want",
"wanna",
"wanting",
"wants",
"let",
"lets",
"letting",
"let's",
"suppose",
"supposing",
"supposes",
"supposed",
"seem",
"seeming",
"seems",
"seemed",
"say",
"saying",
"says",
"said",
"know",
```

```
"knowing",
"knows",
"knew",
"known",
"look",
"looking",
"looked",
"think",
"thinking",
"thinks",
"thought",
"feel",
"feels",
"felt",
"based",
"put",
"puts",
#"wanted"   #Omitted since the advective is relevant
# Question words and associated words
"who",
"who's",
"who've",
"who'd",
"whoever",
"whoever's",
"whom",
"whomever",
"whomever's",
"whose",
"whosever",
"whosever's",
"when",
"whenever",
"which",
"whichever",
"where",
"where's",
"where'd",
"wherever",
"why",
"why's",
"why'd",
"whyever",
"what",
"what's",
"whatever",
"whence",
```

```
"how",
"how's",
"how'd",
"however",
"whether",
"whatsoever",
# Connector words and irrelevant adverbs
"and",
"or",
"not",
"because",
"also",
"always",
"never",
"only",
"really",
"very",
"greatly",
"extremely",
"somewhat",
"no",
"nope",
"nah",
"yes",
"yep",
"yeh",
"yeah",
"maybe",
"perhaps",
"more",
"most",
"less",
"least",
"good",
"great",
"well",
"better",
"best",
"bad",
"worse",
"worst",
"too",
"thru",
"though",
"although",
"yet",
"already",
```

```
"then",
"even",
"now",
"sometimes",
"still",
"together",
"altogether",
"entirely",
"fully",
"entire",
"whole",
"completely",
"utterly",
"seemingly",
"apparently",
"clearly",
"obviously",
"actually",
"actual",
"usually",
"usual",
"literally",
"honestly",
"absolutely",
"definitely",
"generally",
"totally",
"finally",
"basically",
"essentially",
"fundamentally",
"automatically",
"immediately",
"necessarily",
"primarily",
"normally",
"perfectly",
"constantly",
"particularly",
"eventually",
"hopefully",
"mainly",
"typically",
"specifically",
"differently",
"appropriately",
"plenty",
```

```
"certainly",
"unfortunately",
"ultimately",
"unlikely",
"likely",
"potentially",
"fortunately",
"personally",
"directly",
"indirectly",
"nearly",
"closely",
"slightly",
"probably",
"possibly",
"especially",
"frequently",
"often",
"oftentimes",
"seldom",
"rarely",
"sure",
"while",
"whilst",
"able",
"unable",
"else",
"ever",
"once",
"twice",
"thrice",
"almost",
"again",
"instead",
"next",
"previous",
"unless",
"somehow",
"anyhow",
"anywhere",
"somewhere",
"everywhere",
"nowhere",
"further",
"anymore",
"later",
"ago",
```

```
"ahead",
"just",
"same",
"different",
"big",
"small",
"little",
"tiny",
"large",
"huge",
"pretty",
"mostly",
"anyway",
"anyways",
"otherwise",
"regardless",
"throughout",
"additionally",
"moreover",
"furthermore",
"meanwhile",
"afterwards",
# Irrelevant nouns
"thing",
"thing's",
"things",
"stuff",
"other's",
"others",
"another's",
"total",
"",
"false",
"none",
"way",
"kind",
# Lettered numbers and order
"zero",
"zeros",
"zeroes",
"one",
"ones",
"two",
"three",
"four",
"five",
"six",
```

```
"seven",
"eight",
"nine",
"ten",
"twenty",
"thirty",
"forty",
"fifty",
"sixty",
"seventy",
"eighty",
"ninety",
"hundred",
"hundreds",
"thousand",
"thousands",
"million",
"millions",
"first",
"last",
"second",
"third",
"fourth",
"fifth",
"sixth",
"seventh",
"eigth",
"ninth",
"tenth",
"firstly",
"secondly",
"thirdly",
"lastly",
# Greetings and slang
"hello",
"hi",
"hey",
"sup",
"yo",
"greetings",
"please",
"okay",
"ok",
"y'all",
"lol",
"rofl",
"thank",
```

```
"thanks",
"alright",
"kinda",
"dont",
"sorry",
"idk",
"tldr",
"tl",
"dr",  #This means that dr (doctor) is a bad feature because of tl;dr
"tbh",
"dude",
"tho",
"aka",
"plz",
"pls",
"bit",
"don",
# Miscellaneous
"www",
"https",
"http",
"com",
"etc"
"html",
"reddit",
"subreddit",
"subreddits",
"comments",
"reply",
"replies",
"thread",
"threads",
"post",
"posts",
"website",
"websites",
"web site",
"web sites"]
print('length custom:',len(stop_words_custom))
```

```
length custom: 589
```

[ ]:
```
#testing stop words
t_start = time.time()

pipe_params = {
    'clf__criterion': ['gini', 'entropy'],
```

```
    'vect__stop_words': [stop_words_library],
    'clf__max_depth': [10, 50, 100, None],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],

}

vectorizer = CountVectorizer()
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
The best accuracy is 88.305.
The winning parameters are {'clf__criterion': 'entropy', 'clf__max_depth': 100,
'clf__min_samples_leaf': 1, 'clf__min_samples_split': 2, 'vect__stop_words':
['to', 'made', 'his', 'sometime', 'those', 'except', 'no', 'which', 'shan',
'detail', 'when', 'con', 'haven', "needn't", 'take', 'ever', 'would', "shan't",
'side', 'thick', 'thereby', 'hence', 'third', 'ours', 'six', 'she', 'then',
'seeming', 'therein', 'front', 'show', 'below', 'amoungst', 'none', "couldn't",
'keep', 'bottom', 'hereby', 'wherein', 'latterly', 'we', 'upon', 'must', 'once',
'more', 'therefore', 'doesn', 'same', 'seems', 'sincere', 'last', 'whence',
'inc', 'about', "she's", 'needn', 'just', 'somewhere', "don't", 'anything',
'empty', 'via', 'often', 'me', 'whole', 'together', 'call', 'whoever',
'everything', 'former', 'am', 'others', 'elsewhere', 'may', 'thereafter',
"you've", 'seem', "wouldn't", 'himself', 'both', 'ltd', 'because', 'hasnt',
'hereupon', 'even', 'herein', 'could', 'among', 'part', 'should', 'system',
'hadn', 'under', 'yourselves', "wasn't", 'fill', 'who', 'a', 'mill', 'whose',
'whether', 'nothing', 'hereafter', 'any', 'are', 'here', 'nor', 'anyhow',
"didn't", 'ma', 'as', 'before', 'd', 'many', 'cant', 'own', 'whereafter',
'otherwise', 'not', 'my', 'hundred', 'against', 'what', 'two', 'twelve', 'onto',
'eg', 'serious', "you're", 'of', 'well', 'the', 'us', 'your', 'he', 'thereupon',
```

```
'along', "hadn't", 'etc', 'whereby', 'been', 'from', 'theirs', 've', 'mustn',
'though', 'ten', 'in', 'never', "you'd", 'least', 'having', 'every', 'with',
'without', 'beforehand', 'i', 'across', 'full', 'couldnt', 'somehow', 'again',
'how', 'becomes', 'you', 'four', 'five', 'until', 'y', 'might', 'll', 'mostly',
'co', 'beside', 'mightn', 'wasn', "mightn't", 'other', 'already', 'her', "it's",
"isn't", 'always', 'by', 'indeed', 'around', 'either', 'noone', 'although',
'beyond', 'or', 'yours', 'is', 'cannot', 'shouldn', 'cry', 'o', 'for', 'only',
'mine', 'out', 'isn', 'less', "mustn't", 'fire', 'fifteen', 'nowhere', 'into',
'nevertheless', 'seemed', 'moreover', 'now', "won't", 'each', 'why', 'eleven',
'at', 'very', 'besides', 'amongst', 'didn', 'don', 'couldn', 'describe',
'ourselves', 'be', 'thin', 'such', 'during', 'someone', "you'll", 'on',
'thence', 'herself', 'rather', 'twenty', 'get', 'toward', 'everyone', 'won',
'too', 'else', 'wouldn', 'if', 'find', 'also', 'eight', 'whereas', "that'll",
'bill', 'itself', 'since', 'sometimes', 'these', 'this', 'off', 'interest',
'where', 'above', 'alone', 'up', "doesn't", 'hers', 'some', 'it', "shouldn't",
'un', 'have', 'anyone', 'please', 'fifty', "weren't", 'all', 'neither', 'they',
'afterwards', "aren't", 'wherever', 'becoming', 'and', 'whereupon', 's', 'name',
'that', 'something', 'has', 'enough', 'further', 'an', 'meanwhile', 'will',
'next', 'thru', 'another', 'perhaps', 'still', 'forty', 'one', 'whatever',
'doing', 'being', 'several', 'sixty', 'everywhere', 'move', 'yourself', 'per',
'whither', 'put', 'give', 'nobody', 'than', 'however', 'aren', 'through',
'namely', 'can', 'almost', 'latter', 'themselves', 'was', 'anywhere', 'there',
'behind', 'formerly', 'its', 'three', 'much', 'nine', 're', 'does', 'back',
'most', 'between', 'within', 'down', 'de', 'over', 'anyway', 'their', 'were',
'amount', 'weren', "should've", 'while', 'towards', 'ain', 'few', 'hasn', 'but',
'become', 'throughout', 't', 'whenever', 'had', 'top', 'ie', 'myself', 'so',
'see', "haven't", 'yet', 'done', 'after', 'whom', 'first', 'them', "hasn't",
'go', 'found', 'due', 'became', 'm', 'did', 'him', 'do', 'thus', 'our']}
Run time: 30.90384078025818 seconds
```

[ ]:
```python
#testing features
t_start = time.time()

pipe_params = {
    'clf__criterion': ['gini', 'entropy'],
    'vect__stop_words': [stop_words_library],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = DecisionTreeClassifier()

pipe = Pipeline(
```

```
    [("vect", vectorizer),("selecter", selecter),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
The best accuracy is 88.719.
The winning parameters are {'clf__criterion': 'gini', 'clf__max_depth': 100,
'clf__min_samples_leaf': 1, 'clf__min_samples_split': 2, 'selecter__k': 3000,
'vect__stop_words': ['to', 'made', 'his', 'sometime', 'those', 'except', 'no',
'which', 'shan', 'detail', 'when', 'con', 'haven', "needn't", 'take', 'ever',
'would', "shan't", 'side', 'thick', 'thereby', 'hence', 'third', 'ours', 'six',
'she', 'then', 'seeming', 'therein', 'front', 'show', 'below', 'amoungst',
'none', "couldn't", 'keep', 'bottom', 'hereby', 'wherein', 'latterly', 'we',
'upon', 'must', 'once', 'more', 'therefore', 'doesn', 'same', 'seems',
'sincere', 'last', 'whence', 'inc', 'about', "she's", 'needn', 'just',
'somewhere', "don't", 'anything', 'empty', 'via', 'often', 'me', 'whole',
'together', 'call', 'whoever', 'everything', 'former', 'am', 'others',
'elsewhere', 'may', 'thereafter', "you've", 'seem', "wouldn't", 'himself',
'both', 'ltd', 'because', 'hasnt', 'hereupon', 'even', 'herein', 'could',
'among', 'part', 'should', 'system', 'hadn', 'under', 'yourselves', "wasn't",
'fill', 'who', 'a', 'mill', 'whose', 'whether', 'nothing', 'hereafter', 'any',
'are', 'here', 'nor', 'anyhow', "didn't", 'ma', 'as', 'before', 'd', 'many',
'cant', 'own', 'whereafter', 'otherwise', 'not', 'my', 'hundred', 'against',
'what', 'two', 'twelve', 'onto', 'eg', 'serious', "you're", 'of', 'well', 'the',
'us', 'your', 'he', 'thereupon', 'along', "hadn't", 'etc', 'whereby', 'been',
'from', 'theirs', 've', 'mustn', 'though', 'ten', 'in', 'never', "you'd",
'least', 'having', 'every', 'with', 'without', 'beforehand', 'i', 'across',
'full', 'couldnt', 'somehow', 'again', 'how', 'becomes', 'you', 'four', 'five',
'until', 'y', 'might', 'll', 'mostly', 'co', 'beside', 'mightn', 'wasn',
"mightn't", 'other', 'already', 'her', "it's", "isn't", 'always', 'by',
'indeed', 'around', 'either', 'noone', 'although', 'beyond', 'or', 'yours',
'is', 'cannot', 'shouldn', 'cry', 'o', 'for', 'only', 'mine', 'out', 'isn',
'less', "mustn't", 'fire', 'fifteen', 'nowhere', 'into', 'nevertheless',
'seemed', 'moreover', 'now', "won't", 'each', 'why', 'eleven', 'at', 'very',
'besides', 'amongst', 'didn', 'don', 'couldn', 'describe', 'ourselves', 'be',
```

```
'thin', 'such', 'during', 'someone', "you'll", 'on', 'thence', 'herself',
'rather', 'twenty', 'get', 'toward', 'everyone', 'won', 'too', 'else', 'wouldn',
'if', 'find', 'also', 'eight', 'whereas', "that'll", 'bill', 'itself', 'since',
'sometimes', 'these', 'this', 'off', 'interest', 'where', 'above', 'alone',
'up', "doesn't", 'hers', 'some', 'it', "shouldn't", 'un', 'have', 'anyone',
'please', 'fifty', "weren't", 'all', 'neither', 'they', 'afterwards', "aren't",
'wherever', 'becoming', 'and', 'whereupon', 's', 'name', 'that', 'something',
'has', 'enough', 'further', 'an', 'meanwhile', 'will', 'next', 'thru',
'another', 'perhaps', 'still', 'forty', 'one', 'whatever', 'doing', 'being',
'several', 'sixty', 'everywhere', 'move', 'yourself', 'per', 'whither', 'put',
'give', 'nobody', 'than', 'however', 'aren', 'through', 'namely', 'can',
'almost', 'latter', 'themselves', 'was', 'anywhere', 'there', 'behind',
'formerly', 'its', 'three', 'much', 'nine', 're', 'does', 'back', 'most',
'between', 'within', 'down', 'de', 'over', 'anyway', 'their', 'were', 'amount',
'weren', "should've", 'while', 'towards', 'ain', 'few', 'hasn', 'but', 'become',
'throughout', 't', 'whenever', 'had', 'top', 'ie', 'myself', 'so', 'see',
"haven't", 'yet', 'done', 'after', 'whom', 'first', 'them', "hasn't", 'go',
'found', 'due', 'became', 'm', 'did', 'him', 'do', 'thus', 'our']}
Run time: 15.338690280914307 seconds
```

[ ]:
```python
#stem lemmatizer
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

class LemmaTokenizer_Pos:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in␣
 ↪word_tokenize(doc) if t.isalpha()]

class LemmaTokenizer:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) if t.
 ↪isalpha()]

class LemmaTokenizer_word:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
```

```
    def __call__(self, doc):
      return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) ]


class StemTokenizer:
    def __init__(self):
      self.wnl =PorterStemmer()
    def __call__(self, doc):
      return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
```

[ ]:
```
#testing lemma => slight improvement
t_start = time.time()

pipe_params = {
    'clf__criterion': ['entropy'],
    'vect__stop_words': [stop_words_library],
    'vect__tokenizer': [LemmaTokenizer_word()],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("selecter", selecter),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:528:
UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is
```

```
not None'
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'ll", "'re", "'s", "'ve",
'make', "n't", 'need', 'sha', 'win', 'wo'] not in stop_words.
  warnings.warn(

The best accuracy is 86.073.
The winning parameters are {'clf__criterion': 'entropy', 'clf__max_depth': 100,
'clf__min_samples_leaf': 1, 'clf__min_samples_split': 5, 'selecter__k': 5000,
'vect__stop_words': ['to', 'made', 'his', 'sometime', 'those', 'except', 'no',
'which', 'shan', 'detail', 'when', 'con', 'haven', "needn't", 'take', 'ever',
'would', "shan't", 'side', 'thick', 'thereby', 'hence', 'third', 'ours', 'six',
'she', 'then', 'seeming', 'therein', 'front', 'show', 'below', 'amoungst',
'none', "couldn't", 'keep', 'bottom', 'hereby', 'wherein', 'latterly', 'we',
'upon', 'must', 'once', 'more', 'therefore', 'doesn', 'same', 'seems',
'sincere', 'last', 'whence', 'inc', 'about', "she's", 'needn', 'just',
'somewhere', "don't", 'anything', 'empty', 'via', 'often', 'me', 'whole',
'together', 'call', 'whoever', 'everything', 'former', 'am', 'others',
'elsewhere', 'may', 'thereafter', "you've", 'seem', "wouldn't", 'himself',
'both', 'ltd', 'because', 'hasnt', 'hereupon', 'even', 'herein', 'could',
'among', 'part', 'should', 'system', 'hadn', 'under', 'yourselves', "wasn't",
'fill', 'who', 'a', 'mill', 'whose', 'whether', 'nothing', 'hereafter', 'any',
'are', 'here', 'nor', 'anyhow', "didn't", 'ma', 'as', 'before', 'd', 'many',
'cant', 'own', 'whereafter', 'otherwise', 'not', 'my', 'hundred', 'against',
'what', 'two', 'twelve', 'onto', 'eg', 'serious', "you're", 'of', 'well', 'the',
'us', 'your', 'he', 'thereupon', 'along', "hadn't", 'etc', 'whereby', 'been',
'from', 'theirs', 've', 'mustn', 'though', 'ten', 'in', 'never', "you'd",
'least', 'having', 'every', 'with', 'without', 'beforehand', 'i', 'across',
'full', 'couldnt', 'somehow', 'again', 'how', 'becomes', 'you', 'four', 'five',
'until', 'y', 'might', 'll', 'mostly', 'co', 'beside', 'mightn', 'wasn',
"mightn't", 'other', 'already', 'her', "it's", "isn't", 'always', 'by',
'indeed', 'around', 'either', 'noone', 'although', 'beyond', 'or', 'yours',
'is', 'cannot', 'shouldn', 'cry', 'o', 'for', 'only', 'mine', 'out', 'isn',
'less', "mustn't", 'fire', 'fifteen', 'nowhere', 'into', 'nevertheless',
'seemed', 'moreover', 'now', "won't", 'each', 'why', 'eleven', 'at', 'very',
'besides', 'amongst', 'didn', 'don', 'couldn', 'describe', 'ourselves', 'be',
'thin', 'such', 'during', 'someone', "you'll", 'on', 'thence', 'herself',
'rather', 'twenty', 'get', 'toward', 'everyone', 'won', 'too', 'else', 'wouldn',
'if', 'find', 'also', 'eight', 'whereas', "that'll", 'bill', 'itself', 'since',
'sometimes', 'these', 'this', 'off', 'interest', 'where', 'above', 'alone',
'up', "doesn't", 'hers', 'some', 'it', "shouldn't", 'un', 'have', 'anyone',
'please', 'fifty', "weren't", 'all', 'neither', 'they', 'afterwards', "aren't",
'wherever', 'becoming', 'and', 'whereupon', 's', 'name', 'that', 'something',
'has', 'enough', 'further', 'an', 'meanwhile', 'will', 'next', 'thru',
'another', 'perhaps', 'still', 'forty', 'one', 'whatever', 'doing', 'being',
'several', 'sixty', 'everywhere', 'move', 'yourself', 'per', 'whither', 'put',
```

```
'give', 'nobody', 'than', 'however', 'aren', 'through', 'namely', 'can',
'almost', 'latter', 'themselves', 'was', 'anywhere', 'there', 'behind',
'formerly', 'its', 'three', 'much', 'nine', 're', 'does', 'back', 'most',
'between', 'within', 'down', 'de', 'over', 'anyway', 'their', 'were', 'amount',
'weren', "should've", 'while', 'towards', 'ain', 'few', 'hasn', 'but', 'become',
'throughout', 't', 'whenever', 'had', 'top', 'ie', 'myself', 'so', 'see',
"haven't", 'yet', 'done', 'after', 'whom', 'first', 'them', "hasn't", 'go',
'found', 'due', 'became', 'm', 'did', 'him', 'do', 'thus', 'our'],
'vect__tokenizer': <__main__.LemmaTokenizer_word object at 0x7f6d73ed0b80>}
Run time: 81.05935955047607 seconds
```

[ ]:
```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    return text
```

[ ]:
```
#testing preprocessor for lowering words and removing numeric values => slight␣
 ↪improvement
t_start = time.time()

pipe_params = {
    'clf__criterion': ['entropy'],
    'vect__stop_words': [stop_words_library],
    'vect__tokenizer': [LemmaTokenizer_word()],
    'vect__preprocessor': [preprocess_text],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("selecter", selecter),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)
```

```
print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:528:
UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is
not None'
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'ll", "'re", "'s", "'ve",
'make', "n't", 'need', 'sha', 'win', 'wo'] not in stop_words.
  warnings.warn(
```

```
The best accuracy is 86.632.
The winning parameters are {'clf__criterion': 'entropy', 'clf__max_depth': 100,
'clf__min_samples_leaf': 2, 'clf__min_samples_split': 5, 'selecter__k': 3000,
'vect__preprocessor': <function preprocess_text at 0x7f6d6a362670>,
'vect__stop_words': ['to', 'made', 'his', 'sometime', 'those', 'except', 'no',
'which', 'shan', 'detail', 'when', 'con', 'haven', "needn't", 'take', 'ever',
'would', "shan't", 'side', 'thick', 'thereby', 'hence', 'third', 'ours', 'six',
'she', 'then', 'seeming', 'therein', 'front', 'show', 'below', 'amoungst',
'none', "couldn't", 'keep', 'bottom', 'hereby', 'wherein', 'latterly', 'we',
'upon', 'must', 'once', 'more', 'therefore', 'doesn', 'same', 'seems',
'sincere', 'last', 'whence', 'inc', 'about', "she's", 'needn', 'just',
'somewhere', "don't", 'anything', 'empty', 'via', 'often', 'me', 'whole',
'together', 'call', 'whoever', 'everything', 'former', 'am', 'others',
'elsewhere', 'may', 'thereafter', "you've", 'seem', "wouldn't", 'himself',
'both', 'ltd', 'because', 'hasnt', 'hereupon', 'even', 'herein', 'could',
'among', 'part', 'should', 'system', 'hadn', 'under', 'yourselves', "wasn't",
'fill', 'who', 'a', 'mill', 'whose', 'whether', 'nothing', 'hereafter', 'any',
'are', 'here', 'nor', 'anyhow', "didn't", 'ma', 'as', 'before', 'd', 'many',
'cant', 'own', 'whereafter', 'otherwise', 'not', 'my', 'hundred', 'against',
'what', 'two', 'twelve', 'onto', 'eg', 'serious', "you're", 'of', 'well', 'the',
'us', 'your', 'he', 'thereupon', 'along', "hadn't", 'etc', 'whereby', 'been',
'from', 'theirs', 've', 'mustn', 'though', 'ten', 'in', 'never', "you'd",
'least', 'having', 'every', 'with', 'without', 'beforehand', 'i', 'across',
'full', 'couldnt', 'somehow', 'again', 'how', 'becomes', 'you', 'four', 'five',
'until', 'y', 'might', 'll', 'mostly', 'co', 'beside', 'mightn', 'wasn',
"mightn't", 'other', 'already', 'her', "it's", "isn't", 'always', 'by',
'indeed', 'around', 'either', 'noone', 'although', 'beyond', 'or', 'yours',
'is', 'cannot', 'shouldn', 'cry', 'o', 'for', 'only', 'mine', 'out', 'isn',
'less', "mustn't", 'fire', 'fifteen', 'nowhere', 'into', 'nevertheless',
'seemed', 'moreover', 'now', "won't", 'each', 'why', 'eleven', 'at', 'very',
'besides', 'amongst', 'didn', 'don', 'couldn', 'describe', 'ourselves', 'be',
```

```
'thin', 'such', 'during', 'someone', "you'll", 'on', 'thence', 'herself',
'rather', 'twenty', 'get', 'toward', 'everyone', 'won', 'too', 'else', 'wouldn',
'if', 'find', 'also', 'eight', 'whereas', "that'll", 'bill', 'itself', 'since',
'sometimes', 'these', 'this', 'off', 'interest', 'where', 'above', 'alone',
'up', "doesn't", 'hers', 'some', 'it', "shouldn't", 'un', 'have', 'anyone',
'please', 'fifty', "weren't", 'all', 'neither', 'they', 'afterwards', "aren't",
'wherever', 'becoming', 'and', 'whereupon', 's', 'name', 'that', 'something',
'has', 'enough', 'further', 'an', 'meanwhile', 'will', 'next', 'thru',
'another', 'perhaps', 'still', 'forty', 'one', 'whatever', 'doing', 'being',
'several', 'sixty', 'everywhere', 'move', 'yourself', 'per', 'whither', 'put',
'give', 'nobody', 'than', 'however', 'aren', 'through', 'namely', 'can',
'almost', 'latter', 'themselves', 'was', 'anywhere', 'there', 'behind',
'formerly', 'its', 'three', 'much', 'nine', 're', 'does', 'back', 'most',
'between', 'within', 'down', 'de', 'over', 'anyway', 'their', 'were', 'amount',
'weren', "should've", 'while', 'towards', 'ain', 'few', 'hasn', 'but', 'become',
'throughout', 't', 'whenever', 'had', 'top', 'ie', 'myself', 'so', 'see',
"haven't", 'yet', 'done', 'after', 'whom', 'first', 'them', "hasn't", 'go',
'found', 'due', 'became', 'm', 'did', 'him', 'do', 'thus', 'our'],
'vect__tokenizer': <__main__.LemmaTokenizer_word object at 0x7f6d73fcd1c0>}
Run time: 81.01828145980835 seconds
```

[ ]:
```python
#testing binary in vectorize
t_start = time.time()

pipe_params = {
    'clf__criterion': ['entropy'],
    'vect__stop_words': [stop_words_library],
    'vect__tokenizer': [LemmaTokenizer_word()],
    'vect__binary': [True,False],
    'vect__preprocessor': [preprocess_text],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("selecter", selecter),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)
```

```
t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-23-a3178dcbc5cc> in <module>
     24 grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1,␣
 ↪n_jobs=-1)
     25
---> 26 grid.fit(train_x, train_y)
     27
     28 t_end = time.time()

/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_search.py in␣
 ↪fit(self, X, y, groups, **fit_params)
    872                 return results
    873
--> 874             self._run_search(evaluate_candidates)
    875
    876             # multimetric is determined here because in the case of a␣
 ↪callable

/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_search.py in␣
 ↪_run_search(self, evaluate_candidates)
   1386     def _run_search(self, evaluate_candidates):
   1387         """Search all candidates in param_grid"""
-> 1388         evaluate_candidates(ParameterGrid(self.param_grid))
   1389
   1390

/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_search.py in␣
 ↪evaluate_candidates(candidate_params, cv, more_results)
    819                     )
    820
--> 821                 out = parallel(
    822                     delayed(_fit_and_score)(
    823                         clone(base_estimator),
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/parallel.py in␣
 ↪__call__(self, iterable)
     61             for delayed_func, args, kwargs in iterable
     62         )
---> 63         return super().__call__(iterable_with_config)
     64
     65

/usr/local/lib/python3.8/dist-packages/joblib/parallel.py in __call__(self,␣
 ↪iterable)
   1096
   1097             with self._backend.retrieval_context():
-> 1098                 self.retrieve()
   1099             # Make sure that we get a last message telling us we are do e
   1100             elapsed_time = time.time() - self._start_time

/usr/local/lib/python3.8/dist-packages/joblib/parallel.py in retrieve(self)
    973             try:
    974                 if getattr(self._backend, 'supports_timeout', False):
--> 975                     self._output.extend(job.get(timeout=self.timeout))
    976                 else:
    977                     self._output.extend(job.get())

/usr/local/lib/python3.8/dist-packages/joblib/_parallel_backends.py in␣
 ↪wrap_future_result(future, timeout)
    565         AsyncResults.get from multiprocessing."""
    566         try:
--> 567             return future.result(timeout=timeout)
    568         except CfTimeoutError as e:
    569             raise TimeoutError from e

/usr/lib/python3.8/concurrent/futures/_base.py in result(self, timeout)
    437                     return self.__get_result()
    438
--> 439                 self._condition.wait(timeout)
    440
    441                 if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:

/usr/lib/python3.8/threading.py in wait(self, timeout)
    300         try:    # restore state no matter what (e.g., KeyboardInterrupt
    301             if timeout is None:
--> 302                 waiter.acquire()
    303                 gotit = True
    304             else:

KeyboardInterrupt:
```

[ ]:
```
#testing normalize => not good
t_start = time.time()

pipe_params = {
    'clf__criterion': ['entropy'],
    'vect__stop_words': [stop_words_library],
    'vect__tokenizer': [LemmaTokenizer_word()],
    'vect__binary': [False],
    'vect__preprocessor': [preprocess_text],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000],
    'normalizer__norm': ['l2','l1',None]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
normalizer = Normalizer()
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("normalizer", normalizer),("selecter",␣
 ↪selecter),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

/usr/local/lib/python3.8/dist-
packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:
60 fits failed out of a total of 180.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.
```

```
Below are more details about the failures:
--------------------------------------------------------------------------------
60 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-
packages/sklearn/model_selection/_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/pipeline.py", line 390,
in fit
    Xt = self._fit(X, y, **fit_params_steps)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/pipeline.py", line 348,
in _fit
    X, fitted_transformer = fit_transform_one_cached(
  File "/usr/local/lib/python3.8/dist-packages/joblib/memory.py", line 349, in
__call__
    return self.func(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/pipeline.py", line 893,
in _fit_transform_one
    res = transformer.fit_transform(X, y, **fit_params)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/base.py", line 855, in
fit_transform
    return self.fit(X, y, **fit_params).transform(X)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/preprocessing/_data.py",
line 1955, in transform
    return normalize(X, norm=self.norm, axis=1, copy=copy)
  File "/usr/local/lib/python3.8/dist-packages/sklearn/preprocessing/_data.py",
line 1783, in normalize
    raise ValueError("'%s' is not a supported norm" % norm)
ValueError: 'None' is not a supported norm

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.8/dist-packages/sklearn/model_selection/_search.py:969:
UserWarning: One or more of the test scores are non-finite: [0.83984071
0.84119075 0.8356352  0.8412296         nan        nan
 0.83706294 0.83565462 0.83424631 0.83981158        nan        nan
 0.84123932 0.83981158 0.84120047 0.82591298        nan        nan
 0.85374903 0.83844211 0.83008936 0.8342366         nan        nan
 0.85654623 0.84958236 0.85516706 0.84260878        nan        nan
 0.86213092 0.85933372 0.85652681 0.84677545        nan        nan]
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:396:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'ll", "'re", "'s", "'ve",
'make', "n't", 'need', 'sha', 'win', 'wo'] not in stop_words.
  warnings.warn(

The best accuracy is 86.213.
```

```
The winning parameters are {'clf__criterion': 'entropy', 'clf__max_depth': 100,
'clf__min_samples_leaf': 4, 'clf__min_samples_split': 5, 'normalizer__norm':
'l2', 'selecter__k': 5000, 'vect__binary': False, 'vect__preprocessor':
<function preprocess_text at 0x7f407144cee0>, 'vect__stop_words':
frozenset({'now', 'along', 'empty', 'don', 'yours', 'well', 'll', 'about',
'four', 'top', 'serious', 'yourselves', 'than', 'both', 'due', 'and', 'into',
'her', 'thereby', 've', 'except', 'see', 'i', 'down', 'ourselves', 'as',
'thick', 'must', 'do', 'she', 'my', 'own', 'us', 'thus', 'very', 'of', 'wasn',
'your', 'its', 'he', 'former', 'yet', 'almost', 'wherever', 'any', 'had',
'that', 'an', 'itself', "shan't", 'nine', 'besides', 'some', 'whereafter',
'who', 'haven', 'thence', 'namely', 'would', 'everything', 'others', 'seems',
'ain', 'ma', 'rather', "aren't", 'while', "mightn't", 'needn', "you'll",
'beyond', "wouldn't", 'five', 'them', 'thru', 'several', 'two', 'name',
'bottom', "couldn't", 'per', 'most', "doesn't", 'ltd', 'give', "wasn't",
'without', 'get', 'ten', "don't", 'couldn', 'hasn', 'made', 'or', "weren't",
"hadn't", 'how', 'found', 'anyhow', 'against', 'myself', 'to', 'always',
"won't", 'here', 'has', 'co', 'around', 'does', "you're", 'before', 'sincere',
'anything', "you'd", 'becomes', 'their', 'hereupon', 'hadn', 'inc', 'having',
'whoever', 'until', 'within', 'd', 'because', 'above', 'part', 'we', 'm',
'afterwards', "mustn't", 'hundred', 'perhaps', 'via', 'three', 'mine', 'where',
'nowhere', 'few', 'thereupon', 'upon', 'whole', 'then', 'somewhere', 'less',
'for', 'often', 'ever', 'amount', 'neither', 'front', "it's", 'these', 'onto',
'they', 'meanwhile', 'twelve', 'all', 'being', 'last', 'towards', 'below',
'many', 'six', 'o', 'seeming', 'throughout', 'together', 'again', "that'll",
'may', 'un', 'seemed', 'doesn', 'amoungst', 'con', 'anyone', 'each', 'shan',
'forty', 'am', 'across', 'over', 'everyone', 'this', 'hence', 'herein', 'full',
'fifteen', 'so', 'least', 'only', 'another', 'third', 'please', 'thereafter',
'sometimes', 'there', 'never', 'can', 'nevertheless', 'when', 'whereupon',
'him', 'not', 'such', 'next', 'those', 'why', 'himself', 'could', 'same',
'should', 'shouldn', 'our', 're', "didn't", 'just', 'back', 'first', 'alone',
'since', 'hers', 'still', 'whenever', 'won', 'anywhere', 'further', 'seem',
'during', 'thin', 'might', "should've", 'was', 'even', 'move', 'fire', 'bill',
'been', 's', 'up', 'at', 'whereas', 'will', 'too', 'eleven', 'mill', 'system',
'whom', 'noone', 'out', 'which', 'but', 'hereafter', 'among', 'cant', 'either',
'nobody', "she's", 'eight', 'indeed', "needn't", 'cry', 'a', 'nothing', 'on',
'also', 'ie', 'find', 'keep', 'themselves', "haven't", 'formerly', 'though',
'someone', 'behind', 'twenty', 'everywhere', 'whose', 'wouldn', "you've",
'therefore', 'be', 'cannot', 'were', 'none', 'one', 'aren', 'mustn', 'whereby',
'through', "hasn't", 'enough', 'once', 'mostly', 'much', 'although', 'his',
'me', 'become', 'amongst', 'the', "isn't", 'done', 'latter', 'you', 'nor',
'whence', 'isn', 'if', 'between', 'every', 'couldnt', 'yourself', 'what',
'weren', 'therein', 'de', 'mightn', 'more', 'ours', 'became', 'eg', 'take',
'have', 'latterly', 'go', 'etc', 'already', 'with', 'wherein', 'from', 'other',
'herself', "shouldn't", 'beforehand', 'call', 'off', 'beside', 'whether',
'sixty', 'somehow', 'in', 'fifty', 'otherwise', 'whatever', 'toward', 'did',
'elsewhere', 'didn', 't', 'sometime', 'hereby', 'moreover', 'show', 'detail',
'no', 'hasnt', 'however', 'side', 'anyway', 'theirs', 'is', 'put', 'interest',
'it', 'by', 'else', 'y', 'whither', 'after', 'fill', 'becoming', 'describe',
```

```
'are', 'doing', 'something', 'under'}), 'vect__tokenizer':
<__main__.LemmaTokenizer_word object at 0x7f40709b1430>}
Run time: 137.79279041290283 seconds
```

[ ]:
```
#testing tfidf  => not good
t_start = time.time()

pipe_params = {
    'clf__criterion': ['entropy'],
    'vect__stop_words': [stop_words_library],
    #'vect__tokenizer': [LemmaTokenizer_word()],
    'vect__binary': [False],
    'vect__preprocessor': [preprocess_text],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000]
}

vectorizer = TfidfVectorizer()
normalizer = Normalizer()
selecter = SelectKBest(chi2)
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("normalizer", normalizer),("selecter",␣
 ↪selecter),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
The best accuracy is 87.888.
The winning parameters are {'clf__criterion': 'entropy', 'clf__max_depth': 100,
'clf__min_samples_leaf': 1, 'clf__min_samples_split': 2, 'selecter__k': 3000,
'vect__binary': False, 'vect__preprocessor': <function preprocess_text at
0x7f6d6a362670>, 'vect__stop_words': ['to', 'made', 'his', 'sometime', 'those',
```

'except', 'no', 'which', 'shan', 'detail', 'when', 'con', 'haven', "needn't",
'take', 'ever', 'would', "shan't", 'side', 'thick', 'thereby', 'hence', 'third',
'ours', 'six', 'she', 'then', 'seeming', 'therein', 'front', 'show', 'below',
'amoungst', 'none', "couldn't", 'keep', 'bottom', 'hereby', 'wherein',
'latterly', 'we', 'upon', 'must', 'once', 'more', 'therefore', 'doesn', 'same',
'seems', 'sincere', 'last', 'whence', 'inc', 'about', "she's", 'needn', 'just',
'somewhere', "don't", 'anything', 'empty', 'via', 'often', 'me', 'whole',
'together', 'call', 'whoever', 'everything', 'former', 'am', 'others',
'elsewhere', 'may', 'thereafter', "you've", 'seem', "wouldn't", 'himself',
'both', 'ltd', 'because', 'hasnt', 'hereupon', 'even', 'herein', 'could',
'among', 'part', 'should', 'system', 'hadn', 'under', 'yourselves', "wasn't",
'fill', 'who', 'a', 'mill', 'whose', 'whether', 'nothing', 'hereafter', 'any',
'are', 'here', 'nor', 'anyhow', "didn't", 'ma', 'as', 'before', 'd', 'many',
'cant', 'own', 'whereafter', 'otherwise', 'not', 'my', 'hundred', 'against',
'what', 'two', 'twelve', 'onto', 'eg', 'serious', "you're", 'of', 'well', 'the',
'us', 'your', 'he', 'thereupon', 'along', "hadn't", 'etc', 'whereby', 'been',
'from', 'theirs', 've', 'mustn', 'though', 'ten', 'in', 'never', "you'd",
'least', 'having', 'every', 'with', 'without', 'beforehand', 'i', 'across',
'full', 'couldnt', 'somehow', 'again', 'how', 'becomes', 'you', 'four', 'five',
'until', 'y', 'might', 'll', 'mostly', 'co', 'beside', 'mightn', 'wasn',
"mightn't", 'other', 'already', 'her', "it's", "isn't", 'always', 'by',
'indeed', 'around', 'either', 'noone', 'although', 'beyond', 'or', 'yours',
'is', 'cannot', 'shouldn', 'cry', 'o', 'for', 'only', 'mine', 'out', 'isn',
'less', "mustn't", 'fire', 'fifteen', 'nowhere', 'into', 'nevertheless',
'seemed', 'moreover', 'now', "won't", 'each', 'why', 'eleven', 'at', 'very',
'besides', 'amongst', 'didn', 'don', 'couldn', 'describe', 'ourselves', 'be',
'thin', 'such', 'during', 'someone', "you'll", 'on', 'thence', 'herself',
'rather', 'twenty', 'get', 'toward', 'everyone', 'won', 'too', 'else', 'wouldn',
'if', 'find', 'also', 'eight', 'whereas', "that'll", 'bill', 'itself', 'since',
'sometimes', 'these', 'this', 'off', 'interest', 'where', 'above', 'alone',
'up', "doesn't", 'hers', 'some', 'it', "shouldn't", 'un', 'have', 'anyone',
'please', 'fifty', "weren't", 'all', 'neither', 'they', 'afterwards', "aren't",
'wherever', 'becoming', 'and', 'whereupon', 's', 'name', 'that', 'something',
'has', 'enough', 'further', 'an', 'meanwhile', 'will', 'next', 'thru',
'another', 'perhaps', 'still', 'forty', 'one', 'whatever', 'doing', 'being',
'several', 'sixty', 'everywhere', 'move', 'yourself', 'per', 'whither', 'put',
'give', 'nobody', 'than', 'however', 'aren', 'through', 'namely', 'can',
'almost', 'latter', 'themselves', 'was', 'anywhere', 'there', 'behind',
'formerly', 'its', 'three', 'much', 'nine', 're', 'does', 'back', 'most',
'between', 'within', 'down', 'de', 'over', 'anyway', 'their', 'were', 'amount',
'weren', "should've", 'while', 'towards', 'ain', 'few', 'hasn', 'but', 'become',
'throughout', 't', 'whenever', 'had', 'top', 'ie', 'myself', 'so', 'see',
"haven't", 'yet', 'done', 'after', 'whom', 'first', 'them', "hasn't", 'go',
'found', 'due', 'became', 'm', 'did', 'him', 'do', 'thus', 'our']}
Run time: 16.844464778900146 seconds

[ ]:
```
#testing stemmization => does not improve
t_start = time.time()

pipe_params = {
    'clf__criterion': ['entropy'],
    'vect__stop_words': [stop_words_library,None],
    'vect__tokenizer': [StemTokenizer()],
    'vect__binary': [False],
    'vect__preprocessor': [preprocess_text],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000],}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("normalizer", normalizer),("selecter",␣
 ↪selecter),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits

/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:396:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['abov', 'afterward', 'alon',
'alreadi', 'alway', 'ani', 'anoth', 'anyon', 'anyth', 'anywher', 'becam',
'becaus', 'becom', 'befor', 'besid', 'cri', 'describ', 'doe', 'dure', 'els',
'elsewher', 'empti', 'everi', 'everyon', 'everyth', 'everywher', 'fifti',
'formerli', 'forti', 'ha', 'henc', 'hereaft', 'herebi', 'hi', 'howev', 'hundr',
'inde', 'latterli', 'mani', 'meanwhil', 'moreov', 'mostli', 'need', 'nobodi',
'noon', 'noth', 'nowher', 'onc', 'onli', 'otherwis', 'ourselv', 'perhap',
'pleas', 'seriou', 'sever', 'sha', 'sinc', 'sincer', 'sixti', 'someon',
```

```
'someth', 'sometim', 'somewher', 'themselv', 'thenc', 'thereaft', 'therebi',
'therefor', 'thi', 'thu', 'togeth', 'twelv', 'twenti', 'veri', 'wa', 'whatev',
'whenc', 'whenev', 'wherea', 'whereaft', 'wherebi', 'wherev', 'whi', 'wo',
'yourselv'] not in stop_words.
  warnings.warn(

The best accuracy is 86.074.
The winning parameters are {'clf__criterion': 'entropy', 'clf__max_depth': 100,
'clf__min_samples_leaf': 2, 'clf__min_samples_split': 2, 'selecter__k': 3000,
'vect__binary': False, 'vect__preprocessor': <function preprocess_text at
0x7f407144cee0>, 'vect__stop_words': frozenset({'now', 'along', 'empty', 'don',
'yours', 'well', 'll', 'about', 'four', 'top', 'serious', 'yourselves', 'than',
'both', 'due', 'and', 'into', 'her', 'thereby', 've', 'except', 'see', 'i',
'down', 'ourselves', 'as', 'thick', 'must', 'do', 'she', 'my', 'own', 'us',
'thus', 'very', 'of', 'wasn', 'your', 'its', 'he', 'former', 'yet', 'almost',
'wherever', 'any', 'had', 'that', 'an', 'itself', "shan't", 'nine', 'besides',
'some', 'whereafter', 'who', 'haven', 'thence', 'namely', 'would', 'everything',
'others', 'seems', 'ain', 'ma', 'rather', "aren't", 'while', "mightn't",
'needn', "you'll", 'beyond', "wouldn't", 'five', 'them', 'thru', 'several',
'two', 'name', 'bottom', "couldn't", 'per', 'most', "doesn't", 'ltd', 'give',
"wasn't", 'without', 'get', 'ten', "don't", 'couldn', 'hasn', 'made', 'or',
"weren't", "hadn't", 'how', 'found', 'anyhow', 'against', 'myself', 'to',
'always', "won't", 'here', 'has', 'co', 'around', 'does', "you're", 'before',
'sincere', 'anything', "you'd", 'becomes', 'their', 'hereupon', 'hadn', 'inc',
'having', 'whoever', 'until', 'within', 'd', 'because', 'above', 'part', 'we',
'm', 'afterwards', "mustn't", 'hundred', 'perhaps', 'via', 'three', 'mine',
'where', 'nowhere', 'few', 'thereupon', 'upon', 'whole', 'then', 'somewhere',
'less', 'for', 'often', 'ever', 'amount', 'neither', 'front', "it's", 'these',
'onto', 'they', 'meanwhile', 'twelve', 'all', 'being', 'last', 'towards',
'below', 'many', 'six', 'o', 'seeming', 'throughout', 'together', 'again',
"that'll", 'may', 'un', 'seemed', 'doesn', 'amoungst', 'con', 'anyone', 'each',
'shan', 'forty', 'am', 'across', 'over', 'everyone', 'this', 'hence', 'herein',
'full', 'fifteen', 'so', 'least', 'only', 'another', 'third', 'please',
'thereafter', 'sometimes', 'there', 'never', 'can', 'nevertheless', 'when',
'whereupon', 'him', 'not', 'such', 'next', 'those', 'why', 'himself', 'could',
'same', 'should', 'shouldn', 'our', 're', "didn't", 'just', 'back', 'first',
'alone', 'since', 'hers', 'still', 'whenever', 'won', 'anywhere', 'further',
'seem', 'during', 'thin', 'might', "should've", 'was', 'even', 'move', 'fire',
'bill', 'been', 's', 'up', 'at', 'whereas', 'will', 'too', 'eleven', 'mill',
'system', 'whom', 'noone', 'out', 'which', 'but', 'hereafter', 'among', 'cant',
'either', 'nobody', "she's", 'eight', 'indeed', "needn't", 'cry', 'a',
'nothing', 'on', 'also', 'ie', 'find', 'keep', 'themselves', "haven't",
'formerly', 'though', 'someone', 'behind', 'twenty', 'everywhere', 'whose',
'wouldn', "you've", 'therefore', 'be', 'cannot', 'were', 'none', 'one', 'aren',
'mustn', 'whereby', 'through', "hasn't", 'enough', 'once', 'mostly', 'much',
'although', 'his', 'me', 'become', 'amongst', 'the', "isn't", 'done', 'latter',
'you', 'nor', 'whence', 'isn', 'if', 'between', 'every', 'couldnt', 'yourself',
'what', 'weren', 'therein', 'de', 'mightn', 'more', 'ours', 'became', 'eg',
```

```
'take', 'have', 'latterly', 'go', 'etc', 'already', 'with', 'wherein', 'from',
'other', 'herself', "shouldn't", 'beforehand', 'call', 'off', 'beside',
'whether', 'sixty', 'somehow', 'in', 'fifty', 'otherwise', 'whatever', 'toward',
'did', 'elsewhere', 'didn', 't', 'sometime', 'hereby', 'moreover', 'show',
'detail', 'no', 'hasnt', 'however', 'side', 'anyway', 'theirs', 'is', 'put',
'interest', 'it', 'by', 'else', 'y', 'whither', 'after', 'fill', 'becoming',
'describe', 'are', 'doing', 'something', 'under'}), 'vect__tokenizer':
<__main__.StemTokenizer object at 0x7f407120e880>}
Run time: 208.76408529281616 seconds
```

[ ]:
```
#testing custom => 86.351.
t_start = time.time()

pipe_params = {
    'clf__criterion': ['entropy'],
    'vect__stop_words': [stop_words_library],
    'vect__tokenizer': [LemmaTokenizer_word()],
    'vect__binary': [False],
    'vect__preprocessor': [preprocess_text],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("selecter", selecter),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:396:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'ll", "'re", "'s", "'ve",
'make', "n't", 'need', 'sha', 'win', 'wo'] not in stop_words.
  warnings.warn(

The best accuracy is 86.351.
The winning parameters are {'clf__criterion': 'entropy', 'clf__max_depth': 100,
'clf__min_samples_leaf': 2, 'clf__min_samples_split': 2, 'selecter__k': 3000,
'vect__binary': False, 'vect__preprocessor': <function preprocess_text at
0x7f407144cee0>, 'vect__stop_words': frozenset({'now', 'along', 'empty', 'don',
'yours', 'well', 'll', 'about', 'four', 'top', 'serious', 'yourselves', 'than',
'both', 'due', 'and', 'into', 'her', 'thereby', 've', 'except', 'see', 'i',
'down', 'ourselves', 'as', 'thick', 'must', 'do', 'she', 'my', 'own', 'us',
'thus', 'very', 'of', 'wasn', 'your', 'its', 'he', 'former', 'yet', 'almost',
'wherever', 'any', 'had', 'that', 'an', 'itself', "shan't", 'nine', 'besides',
'some', 'whereafter', 'who', 'haven', 'thence', 'namely', 'would', 'everything',
'others', 'seems', 'ain', 'ma', 'rather', "aren't", 'while', "mightn't",
'needn', "you'll", 'beyond', "wouldn't", 'five', 'them', 'thru', 'several',
'two', 'name', 'bottom', "couldn't", 'per', 'most', "doesn't", 'ltd', 'give',
"wasn't", 'without', 'get', 'ten', "don't", 'couldn', 'hasn', 'made', 'or',
"weren't", "hadn't", 'how', 'found', 'anyhow', 'against', 'myself', 'to',
'always', "won't", 'here', 'has', 'co', 'around', 'does', "you're", 'before',
'sincere', 'anything', "you'd", 'becomes', 'their', 'hereupon', 'hadn', 'inc',
'having', 'whoever', 'until', 'within', 'd', 'because', 'above', 'part', 'we',
'm', 'afterwards', "mustn't", 'hundred', 'perhaps', 'via', 'three', 'mine',
'where', 'nowhere', 'few', 'thereupon', 'upon', 'whole', 'then', 'somewhere',
'less', 'for', 'often', 'ever', 'amount', 'neither', 'front', "it's", 'these',
'onto', 'they', 'meanwhile', 'twelve', 'all', 'being', 'last', 'towards',
'below', 'many', 'six', 'o', 'seeming', 'throughout', 'together', 'again',
"that'll", 'may', 'un', 'seemed', 'doesn', 'amoungst', 'con', 'anyone', 'each',
'shan', 'forty', 'am', 'across', 'over', 'everyone', 'this', 'hence', 'herein',
'full', 'fifteen', 'so', 'least', 'only', 'another', 'third', 'please',
'thereafter', 'sometimes', 'there', 'never', 'can', 'nevertheless', 'when',
'whereupon', 'him', 'not', 'such', 'next', 'those', 'why', 'himself', 'could',
'same', 'should', 'shouldn', 'our', 're', "didn't", 'just', 'back', 'first',
'alone', 'since', 'hers', 'still', 'whenever', 'won', 'anywhere', 'further',
'seem', 'during', 'thin', 'might', "should've", 'was', 'even', 'move', 'fire',
'bill', 'been', 's', 'up', 'at', 'whereas', 'will', 'too', 'eleven', 'mill',
'system', 'whom', 'noone', 'out', 'which', 'but', 'hereafter', 'among', 'cant',
'either', 'nobody', "she's", 'eight', 'indeed', "needn't", 'cry', 'a',
'nothing', 'on', 'also', 'ie', 'find', 'keep', 'themselves', "haven't",
'formerly', 'though', 'someone', 'behind', 'twenty', 'everywhere', 'whose',
'wouldn', "you've", 'therefore', 'be', 'cannot', 'were', 'none', 'one', 'aren',
'mustn', 'whereby', 'through', "hasn't", 'enough', 'once', 'mostly', 'much',
'although', 'his', 'me', 'become', 'amongst', 'the', "isn't", 'done', 'latter',
'you', 'nor', 'whence', 'isn', 'if', 'between', 'every', 'couldnt', 'yourself',
'what', 'weren', 'therein', 'de', 'mightn', 'more', 'ours', 'became', 'eg',
```

```
'take', 'have', 'latterly', 'go', 'etc', 'already', 'with', 'wherein', 'from',
'other', 'herself', "shouldn't", 'beforehand', 'call', 'off', 'beside',
'whether', 'sixty', 'somehow', 'in', 'fifty', 'otherwise', 'whatever', 'toward',
'did', 'elsewhere', 'didn', 't', 'sometime', 'hereby', 'moreover', 'show',
'detail', 'no', 'hasnt', 'however', 'side', 'anyway', 'theirs', 'is', 'put',
'interest', 'it', 'by', 'else', 'y', 'whither', 'after', 'fill', 'becoming',
'describe', 'are', 'doing', 'something', 'under'}), 'vect__tokenizer':
<__main__.LemmaTokenizer_word object at 0x7f4071487370>}
Run time: 47.835500717163086 seconds
```

[ ]:
```python
#removing custom preprocessor => 86.21
t_start = time.time()

pipe_params = {
    'clf__criterion': ['entropy'],
    'vect__stop_words': [stop_words_library],
    'vect__tokenizer': [LemmaTokenizer_word()],
    'vect__binary': [False],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("selecter", selecter),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:396:
```

```
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'ll", "'re", "'s", "'ve",
'make', "n't", 'need', 'sha', 'win', 'wo'] not in stop_words.
  warnings.warn(

The best accuracy is 86.21.
The winning parameters are {'clf__criterion': 'entropy', 'clf__max_depth': 100,
'clf__min_samples_leaf': 1, 'clf__min_samples_split': 2, 'selecter__k': 5000,
'vect__binary': False, 'vect__stop_words': frozenset({'now', 'along', 'empty',
'don', 'yours', 'well', 'll', 'about', 'four', 'top', 'serious', 'yourselves',
'than', 'both', 'due', 'and', 'into', 'her', 'thereby', 've', 'except', 'see',
'i', 'down', 'ourselves', 'as', 'thick', 'must', 'do', 'she', 'my', 'own', 'us',
'thus', 'very', 'of', 'wasn', 'your', 'its', 'he', 'former', 'yet', 'almost',
'wherever', 'any', 'had', 'that', 'an', 'itself', "shan't", 'nine', 'besides',
'some', 'whereafter', 'who', 'haven', 'thence', 'namely', 'would', 'everything',
'others', 'seems', 'ain', 'ma', 'rather', "aren't", 'while', "mightn't",
'needn', "you'll", 'beyond', "wouldn't", 'five', 'them', 'thru', 'several',
'two', 'name', 'bottom', "couldn't", 'per', 'most', "doesn't", 'ltd', 'give',
"wasn't", 'without', 'get', 'ten', "don't", 'couldn', 'hasn', 'made', 'or',
"weren't", "hadn't", 'how', 'found', 'anyhow', 'against', 'myself', 'to',
'always', "won't", 'here', 'has', 'co', 'around', 'does', "you're", 'before',
'sincere', 'anything', "you'd", 'becomes', 'their', 'hereupon', 'hadn', 'inc',
'having', 'whoever', 'until', 'within', 'd', 'because', 'above', 'part', 'we',
'm', 'afterwards', "mustn't", 'hundred', 'perhaps', 'via', 'three', 'mine',
'where', 'nowhere', 'few', 'thereupon', 'upon', 'whole', 'then', 'somewhere',
'less', 'for', 'often', 'ever', 'amount', 'neither', 'front', "it's", 'these',
'onto', 'they', 'meanwhile', 'twelve', 'all', 'being', 'last', 'towards',
'below', 'many', 'six', 'o', 'seeming', 'throughout', 'together', 'again',
"that'll", 'may', 'un', 'seemed', 'doesn', 'amoungst', 'con', 'anyone', 'each',
'shan', 'forty', 'am', 'across', 'over', 'everyone', 'this', 'hence', 'herein',
'full', 'fifteen', 'so', 'least', 'only', 'another', 'third', 'please',
'thereafter', 'sometimes', 'there', 'never', 'can', 'nevertheless', 'when',
'whereupon', 'him', 'not', 'such', 'next', 'those', 'why', 'himself', 'could',
'same', 'should', 'shouldn', 'our', 're', "didn't", 'just', 'back', 'first',
'alone', 'since', 'hers', 'still', 'whenever', 'won', 'anywhere', 'further',
'seem', 'during', 'thin', 'might', "should've", 'was', 'even', 'move', 'fire',
'bill', 'been', 's', 'up', 'at', 'whereas', 'will', 'too', 'eleven', 'mill',
'system', 'whom', 'noone', 'out', 'which', 'but', 'hereafter', 'among', 'cant',
'either', 'nobody', "she's", 'eight', 'indeed', "needn't", 'cry', 'a',
'nothing', 'on', 'also', 'ie', 'find', 'keep', 'themselves', "haven't",
'formerly', 'though', 'someone', 'behind', 'twenty', 'everywhere', 'whose',
'wouldn', "you've", 'therefore', 'be', 'cannot', 'were', 'none', 'one', 'aren',
'mustn', 'whereby', 'through', "hasn't", 'enough', 'once', 'mostly', 'much',
'although', 'his', 'me', 'become', 'amongst', 'the', "isn't", 'done', 'latter',
'you', 'nor', 'whence', 'isn', 'if', 'between', 'every', 'couldnt', 'yourself',
'what', 'weren', 'therein', 'de', 'mightn', 'more', 'ours', 'became', 'eg',
'take', 'have', 'latterly', 'go', 'etc', 'already', 'with', 'wherein', 'from',
'other', 'herself', "shouldn't", 'beforehand', 'call', 'off', 'beside',
```

```
'whether', 'sixty', 'somehow', 'in', 'fifty', 'otherwise', 'whatever', 'toward',
'did', 'elsewhere', 'didn', 't', 'sometime', 'hereby', 'moreover', 'show',
'detail', 'no', 'hasnt', 'however', 'side', 'anyway', 'theirs', 'is', 'put',
'interest', 'it', 'by', 'else', 'y', 'whither', 'after', 'fill', 'becoming',
'describe', 'are', 'doing', 'something', 'under'}), 'vect__tokenizer':
<__main__.LemmaTokenizer_word object at 0x7f406360e400>}
Run time: 46.78005290031433 seconds
```

[ ]:
```
#testing Ngram
t_start = time.time()

pipe_params = {
    'clf__criterion': ['entropy'],
    'vect__stop_words': [list(stop_words_custom)],
    'vect__tokenizer': [LemmaTokenizer_word()],
    'vect__binary': [False],
    'vect__ngram_range':[(1,1)],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000],
    "normalizer__norm": ['l2','l1']
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
normalizer = Normalizer()
model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", vectorizer),("normalizer",normalizer),("selecter",␣
 ↪selecter),("clf",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:528:
UserWarning: The parameter 'token_pattern' will not be used since 'tokenizer' is
not None'
  warnings.warn(
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'ll", "'m", "'re", "'s",
"'ve", 'ai', 'base', 'bite', 'ca', 'comment', 'concern', 'consider', 'exclude',
'follow', 'gon', 'greet', 'leave', "n't", 'na', 'regard', 'sit', 'site', 'wan',
'web', 'wo'] not in stop_words.
  warnings.warn(

The best accuracy is 84.408.
The winning parameters are {'clf__criterion': 'entropy', 'clf__max_depth': 100,
'clf__min_samples_leaf': 4, 'clf__min_samples_split': 2, 'normalizer__norm':
'l1', 'selecter__k': 3000, 'vect__binary': False, 'vect__ngram_range': (1, 1),
'vect__stop_words': ['i', "i'll", "i'd", "i'm", "i've", 'ive', 'me', 'myself',
'you', "you'll", "you'd", "you're", "you've", 'yourself', 'he', "he'll", "he'd",
"he's", 'him', 'she', "she'll", "she'd", "she's", 'her', 'it', "it'll", "it'd",
"it's", 'itself', 'oneself', 'we', "we'll", "we'd", "we're", "we've", 'us',
'ourselves', 'they', "they'll", "they'd", "they're", "they've", 'them',
'themselves', 'everyone', "everyone's", 'everybody', "everybody's", 'someone',
"someone's", 'somebody', "somebody's", 'nobody', "nobody's", 'anyone',
"anyone's", 'everything', "everything's", 'something', "something's", 'nothing',
"nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this', 'that',
"that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its', 'our',
'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots', 'some',
'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each', 'every',
'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather', 'quite',
'aboard', 'about', 'above', 'across', 'after', 'against', 'along', 'amid',
'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away', 'before',
'behind', 'below', 'beneath', 'beside', 'besides', 'between', 'beyond', 'but',
'by', 'concerning', 'considering', 'despite', 'down', 'during', 'except',
'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here', "here's",
'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off', 'on',
'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
```

'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites'], 'vect__tokenizer':
<__main__.LemmaTokenizer_word object at 0x7f6d73fe0dc0>}
Run time: 170.62921714782715 seconds

[ ]:
```
#testing features
t_start = time.time()

final_pipe_params = {
    'clf__criterion': ['gini', 'entropy'],
    'vect__stop_words': [list(stop_words_custom)],
    'clf__max_depth': [100],
    'clf__min_samples_split': [2, 5],
    'clf__min_samples_leaf': [1, 2, 4],
    'selecter__k':[5000,3000]
}

final_vectorizer = CountVectorizer()
final_selecter = SelectKBest(chi2)
final_model = DecisionTreeClassifier()

pipe = Pipeline(
    [("vect", final_vectorizer),("selecter",␣
 ↪final_selecter),("clf",final_model)]
)

final_grid = model_selection.GridSearchCV(pipe, final_pipe_params, verbose=1,␣
 ↪n_jobs=-1)

final_grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(final_grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {final_grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
The best accuracy is 88.444.
The winning parameters are {'clf__criterion': 'gini', 'clf__max_depth': 100,
'clf__min_samples_leaf': 1, 'clf__min_samples_split': 5, 'selecter__k': 5000,
'vect__stop_words': ['i', "i'll", "i'd", "i'm", "i've", 'ive', 'me', 'myself',
'you', "you'll", "you'd", "you're", "you've", 'yourself', 'he', "he'll", "he'd",
"he's", 'him', 'she', "she'll", "she'd", "she's", 'her', 'it', "it'll", "it'd",
"it's", 'itself', 'oneself', 'we', "we'll", "we'd", "we're", "we've", 'us',
'ourselves', 'they', "they'll", "they'd", "they're", "they've", 'them',
'themselves', 'everyone', "everyone's", 'everybody', "everybody's", 'someone',
"someone's", 'somebody', "somebody's", 'nobody', "nobody's", 'anyone',
"anyone's", 'everything', "everything's", 'something', "something's", 'nothing',
```

"nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this', 'that',
"that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its', 'our',
'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots', 'some',
'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each', 'every',
'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather', 'quite',
'aboard', 'about', 'above', 'across', 'after', 'against', 'along', 'amid',
'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away', 'before',
'behind', 'below', 'beneath', 'beside', 'besides', 'between', 'beyond', 'but',
'by', 'concerning', 'considering', 'despite', 'down', 'during', 'except',
'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here', "here's",
'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off', 'on',
'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',

```
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites']}
Run time: 14.492570400238037 seconds

/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(
```

[ ]:
```
print(round(final_grid.best_score_ * 100,3))
print(f"Run time: {elapsed_time} seconds")
y_pred = final_grid.predict(test_x)
create_test_csv(y_pred,"DesicionTree_04032023_02.csv")
```

```
88.444
Run time: 14.492570400238037 seconds
File saved.
```

[ ]:
```
def print_best_params(grid):
  bestParameters = grid.best_estimator_.get_params()
  # print(bestParameters)
  for paramName in sorted(bestParameters.keys()):
    print("\t%s: %r" % (paramName, bestParameters[paramName]))
```

[ ]:
```
print_best_params(final_grid)
```

```
        clf: DecisionTreeClassifier(max_depth=100, min_samples_split=5)
        clf__ccp_alpha: 0.0
        clf__class_weight: None
```

```
        clf__criterion: 'gini'
        clf__max_depth: 100
        clf__max_features: None
        clf__max_leaf_nodes: None
        clf__min_impurity_decrease: 0.0
        clf__min_samples_leaf: 1
        clf__min_samples_split: 5
        clf__min_weight_fraction_leaf: 0.0
        clf__random_state: None
        clf__splitter: 'best'
        memory: None
        selecter: SelectKBest(k=5000, score_func=<function chi2 at
0x7f6d76ec2b80>)
        selecter__k: 5000
        selecter__score_func: <function chi2 at 0x7f6d76ec2b80>
        steps: [('vect', CountVectorizer(stop_words=['i', "i'll", "i'd", "i'm",
"i've", 'ive', 'me',
                                'myself', 'you', "you'll", "you'd", "you're",
                                "you've", 'yourself', 'he', "he'll", "he'd", "he's",
                                'him', 'she', "she'll", "she'd", "she's", 'her',
                                'it', "it'll", "it'd", "it's", 'itself', 'oneself',
…])), ('selecter', SelectKBest(k=5000, score_func=<function chi2 at
0x7f6d76ec2b80>)), ('clf', DecisionTreeClassifier(max_depth=100,
min_samples_split=5))]
        vect: CountVectorizer(stop_words=['i', "i'll", "i'd", "i'm", "i've",
'ive', 'me',
                                'myself', 'you', "you'll", "you'd", "you're",
                                "you've", 'yourself', 'he', "he'll", "he'd", "he's",
                                'him', 'she', "she'll", "she'd", "she's", 'her',
                                'it', "it'll", "it'd", "it's", 'itself', 'oneself',
…])
        vect__analyzer: 'word'
        vect__binary: False
        vect__decode_error: 'strict'
        vect__dtype: <class 'numpy.int64'>
        vect__encoding: 'utf-8'
        vect__input: 'content'
        vect__lowercase: True
        vect__max_df: 1.0
        vect__max_features: None
        vect__min_df: 1
        vect__ngram_range: (1, 1)
        vect__preprocessor: None
        vect__stop_words: ['i', "i'll", "i'd", "i'm", "i've", 'ive', 'me',
'myself', 'you', "you'll", "you'd", "you're", "you've", 'yourself', 'he',
"he'll", "he'd", "he's", 'him', 'she', "she'll", "she'd", "she's", 'her', 'it',
"it'll", "it'd", "it's", 'itself', 'oneself', 'we', "we'll", "we'd", "we're",
"we've", 'us', 'ourselves', 'they', "they'll", "they'd", "they're", "they've",
```

'them', 'themselves', 'everyone', "everyone's", 'everybody', "everybody's",
'someone', "someone's", 'somebody', "somebody's", 'nobody', "nobody's",
'anyone', "anyone's", 'everything', "everything's", 'something', "something's",
'nothing', "nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this',
'that', "that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its',
'our', 'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots',
'some', 'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each',
'every', 'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather',
'quite', 'aboard', 'about', 'above', 'across', 'after', 'against', 'along',
'amid', 'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away',
'before', 'behind', 'below', 'beneath', 'beside', 'besides', 'between',
'beyond', 'but', 'by', 'concerning', 'considering', 'despite', 'down', 'during',
'except', 'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here',
"here's", 'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off',
'on', 'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',

```
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites']
       vect__strip_accents: None
       vect__token_pattern: '(?u)\\b\\w\\w+\\b'
       vect__tokenizer: None
       vect__vocabulary: None
       verbose: False
```

[ ]:
```
# Step 5: Make predictions on test data using the trained model
```

[ ]:
```
############################################################ final
```

final_random_forest

March 12, 2023

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from google.colab import drive
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import random
import time
import re
import string
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2,␣
 ↪f_classif,mutual_info_classif,f_regression
from sklearn.preprocessing import Normalizer
from sklearn import model_selection
from sklearn import svm
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
nltk.download('wordnet')
nltk.download('stopwords')

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[ ]:
```
#import the data
drive.mount('/content/gdrive/', force_remount=True)

train_data_initial = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/train.
 ↪csv')
test_data = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/test.csv')

print('shape train:',train_data_initial.shape)
print('shape test:',test_data.shape)
```

```
Mounted at /content/gdrive/
shape train: (718, 2)
shape test: (279, 2)
```

[ ]:
```
def shuffle_data(df):
    random.seed(0)  # Use a fixed seed for the random number generator
    df = df.sample(frac=1, random_state=0).reset_index(drop=True)
    return df
```

[ ]:
```
#function for creating the test csv file to upload to kaggle
def create_test_csv(data, outfile_name):
  rawdata= {'subreddit':data}
  csv = pd.DataFrame(rawdata, columns = ['subreddit'])
```

```
  csv.to_csv(outfile_name,index=True, header=True)
  print ("File saved.")
```

[ ]:
```
#shuffle the data and split the features from the label
train_data = shuffle_data(train_data_initial)

#train_data = train_data.sample(500).reset_index(drop=True)
#train_data = train_data.head(200)


train_x = train_data["body"]
train_y = train_data["subreddit"]
test_x = test_data["body"]
```

[ ]:
```
print(train_x[5])
```

```
Hi there /u/LakotaPride! Welcome to /r/Trump.  [](/sp)

Thank you for posting on r/Trump Please follow all rules and guidelines. Inform
the mods if you have any concerns. [](/sp) Join our live
[discord](https://discord.gg/kh4Wv9DavE) chat to talk to your fellow patriots!
If you have any issues please reach out.


*I am a bot, and this action was performed automatically. Please [contact the
moderators of this subreddit](/message/compose/?to=/r/trump) if you have any
questions or concerns.*
```

[ ]:
```
#create a dictionary of stop words
stop_words_nltk = set(stopwords.words('english'))
stop_words_sklearn = text.ENGLISH_STOP_WORDS
stop_words_library = stop_words_sklearn.union(stop_words_nltk)

stop_words_custom = [
    # All pronouns and associated words
    "i","i'll","i'd","i'm","i've","ive","me","myself","you",
    "you'll",
    "you'd",
    "you're",
    "you've",
    "yourself",
    "he",
    "he'll",
    "he'd",
    "he's",
    "him",
    "she",
```

```
    "she'll",
    "she'd",
    "she's",
    "her",
    "it",
    "it'll",
    "it'd",
    "it's",
    "itself",
    "oneself",
    "we",
    "we'll",
    "we'd",
    "we're",
    "we've",
    "us",
    "ourselves",
    "they",
    "they'll",
    "they'd",
    "they're",
    "they've",
    "them",
    "themselves",
    "everyone",
    "everyone's",
    "everybody",
    "everybody's",
    "someone",
    "someone's",
    "somebody",
    "somebody's",
    "nobody",
    "nobody's",
    "anyone",
    "anyone's",
    "everything",
    "everything's",
    "something",
    "something's",
    "nothing",
    "nothing's",
    "anything",
    "anything's",
    # All determiners and associated words
    "a",
    "an",
```

```
    "the",
    "this",
    "that",
    "that's",
    "these",
    "those",
    "my",
    #"mine",   #Omitted since mine can refer to something else
    "your",
    "yours",
    "his",
    "hers",
    "its",
    "our",
    "ours",
    "own",
    "their",
    "theirs",
    "few",
    "much",
    "many",
    "lot",
    "lots",
    "some",
    "any",
    "enough",
    "all",
    "both",
    "half",
    "either",
    "neither",
    "each",
    "every",
    "certain",
    "other",
    "another",
    "such",
    "several",
    "multiple",
    # "what",    #Dealt with later on
    "rather",
    "quite",
    # All prepositions
    "aboard",
    "about",
    "above",
    "across",
```

```
    "after",
    "against",
    "along",
    "amid",
    "amidst",
    "among",
    "amongst",
    "anti",
    "around",
    "as",
    "at",
    "away",
    "before",
    "behind",
    "below",
    "beneath",
    "beside",
    "besides",
    "between",
    "beyond",
    "but",
    "by",
    "concerning",
    "considering",
    "despite",
    "down",
    "during",
    "except",
    "excepting",
    "excluding",
    "far",
    "following",
    "for",
    "from",
    "here",
    "here's",
    "in",
    "inside",
    "into",
    "left",
    "like",
    "minus",
    "near",
    "of",
    "off",
    "on",
    "onto",
```

```
    "opposite",
    "out",
    "outside",
    "over",
    "past",
    "per",
    "plus",
    "regarding",
    "right",
    #"round",   #Omitted
    #"save",    #Omitted
    "since",
    "than",
    "there",
    "there's",
    "through",
    "to",
    "toward",
    "towards",
    "under",
    "underneath",
    "unlike",
    "until",
    "up",
    "upon",
    "versus",
    "via",
    "with",
    "within",
    "without",
    # Irrelevant verbs
    "may",
    "might",
    "will",
    "won't",
    "would",
    "wouldn't",
    "can",
    "can't",
    "cannot",
    "could",
    "couldn't",
    "should",
    "shouldn't",
    "must",
    "must've",
    "be",
```

```
    "being",
    "been",
    "am",
    "are",
    "aren't",
    "ain't",
    "is",
    "isn't",
    "was",
    "wasn't",
    "were",
    "weren't",
    "do",
    "doing",
    "don't",
    "does",
    "doesn't",
    "did",
    "didn't",
    "done",
    "have",
    "haven't",
    "having",
    "has",
    "hasn't",
    "had",
    "hadn't",
    "get",
    "getting",
    "gets",
    "got",
    "gotten",
    "go",
    "going",
    "gonna",
    "goes",
    "went",
    "gone",
    "make",
    "making",
    "makes",
    "made",
    "take",
    "taking",
    "takes",
    "took",
    "taken",
```

```
    "need",
    "needing",
    "needs",
    "needed",
    "use",
    "using",
    "uses",
    "used",
    "want",
    "wanna",
    "wanting",
    "wants",
    "let",
    "lets",
    "letting",
    "let's",
    "suppose",
    "supposing",
    "supposes",
    "supposed",
    "seem",
    "seeming",
    "seems",
    "seemed",
    "say",
    "saying",
    "says",
    "said",
    "know",
    "knowing",
    "knows",
    "knew",
    "known",
    "look",
    "looking",
    "looked",
    "think",
    "thinking",
    "thinks",
    "thought",
    "feel",
    "feels",
    "felt",
    "based",
    "put",
    "puts",
    #"wanted"   #Omitted since the advective is relevant
```

```
    # Question words and associated words
    "who",
    "who's",
    "who've",
    "who'd",
    "whoever",
    "whoever's",
    "whom",
    "whomever",
    "whomever's",
    "whose",
    "whosever",
    "whosever's",
    "when",
    "whenever",
    "which",
    "whichever",
    "where",
    "where's",
    "where'd",
    "wherever",
    "why",
    "why's",
    "why'd",
    "whyever",
    "what",
    "what's",
    "whatever",
    "whence",
    "how",
    "how's",
    "how'd",
    "however",
    "whether",
    "whatsoever",
    # Connector words and irrelevant adverbs
    "and",
    "or",
    "not",
    "because",
    "also",
    "always",
    "never",
    "only",
    "really",
    "very",
    "greatly",
```

```
    "extremely",
    "somewhat",
    "no",
    "nope",
    "nah",
    "yes",
    "yep",
    "yeh",
    "yeah",
    "maybe",
    "perhaps",
    "more",
    "most",
    "less",
    "least",
    "good",
    "great",
    "well",
    "better",
    "best",
    "bad",
    "worse",
    "worst",
    "too",
    "thru",
    "though",
    "although",
    "yet",
    "already",
    "then",
    "even",
    "now",
    "sometimes",
    "still",
    "together",
    "altogether",
    "entirely",
    "fully",
    "entire",
    "whole",
    "completely",
    "utterly",
    "seemingly",
    "apparently",
    "clearly",
    "obviously",
    "actually",
```

```
    "actual",
    "usually",
    "usual",
    "literally",
    "honestly",
    "absolutely",
    "definitely",
    "generally",
    "totally",
    "finally",
    "basically",
    "essentially",
    "fundamentally",
    "automatically",
    "immediately",
    "necessarily",
    "primarily",
    "normally",
    "perfectly",
    "constantly",
    "particularly",
    "eventually",
    "hopefully",
    "mainly",
    "typically",
    "specifically",
    "differently",
    "appropriately",
    "plenty",
    "certainly",
    "unfortunately",
    "ultimately",
    "unlikely",
    "likely",
    "potentially",
    "fortunately",
    "personally",
    "directly",
    "indirectly",
    "nearly",
    "closely",
    "slightly",
    "probably",
    "possibly",
    "especially",
    "frequently",
    "often",
```

```
    "oftentimes",
    "seldom",
    "rarely",
    "sure",
    "while",
    "whilst",
    "able",
    "unable",
    "else",
    "ever",
    "once",
    "twice",
    "thrice",
    "almost",
    "again",
    "instead",
    "next",
    "previous",
    "unless",
    "somehow",
    "anyhow",
    "anywhere",
    "somewhere",
    "everywhere",
    "nowhere",
    "further",
    "anymore",
    "later",
    "ago",
    "ahead",
    "just",
    "same",
    "different",
    "big",
    "small",
    "little",
    "tiny",
    "large",
    "huge",
    "pretty",
    "mostly",
    "anyway",
    "anyways",
    "otherwise",
    "regardless",
    "throughout",
    "additionally",
```

```
    "moreover",
    "furthermore",
    "meanwhile",
    "afterwards",
    # Irrelevant nouns
    "thing",
    "thing's",
    "things",
    "stuff",
    "other's",
    "others",
    "another's",
    "total",
    "",
    "false",
    "none",
    "way",
    "kind",
    # Lettered numbers and order
    "zero",
    "zeros",
    "zeroes",
    "one",
    "ones",
    "two",
    "three",
    "four",
    "five",
    "six",
    "seven",
    "eight",
    "nine",
    "ten",
    "twenty",
    "thirty",
    "forty",
    "fifty",
    "sixty",
    "seventy",
    "eighty",
    "ninety",
    "hundred",
    "hundreds",
    "thousand",
    "thousands",
    "million",
    "millions",
```

```
    "first",
    "last",
    "second",
    "third",
    "fourth",
    "fifth",
    "sixth",
    "seventh",
    "eigth",
    "ninth",
    "tenth",
    "firstly",
    "secondly",
    "thirdly",
    "lastly",
    # Greetings and slang
    "hello",
    "hi",
    "hey",
    "sup",
    "yo",
    "greetings",
    "please",
    "okay",
    "ok",
    "y'all",
    "lol",
    "rofl",
    "thank",
    "thanks",
    "alright",
    "kinda",
    "dont",
    "sorry",
    "idk",
    "tldr",
    "tl",
    "dr",  #This means that dr (doctor) is a bad feature because of tl;dr
    "tbh",
    "dude",
    "tho",
    "aka",
    "plz",
    "pls",
    "bit",
    "don",
    # Miscellaneous
```

```
    "www",
    "https",
    "http",
    "com",
    "etc",
    "html",
    "reddit",
    "subreddit",
    "subreddits",
    "comments",
    "reply",
    "replies",
    "thread",
    "threads",
    "post",
    "posts",
    "website",
    "websites",
    "web site",
    "web sites"]
print('length custom:',len(stop_words_custom))
```

```
length custom: 590
```

[ ]:

[ ]:
```
#stem lemmatizer
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

class LemmaTokenizer_Pos:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in␣
 ↪word_tokenize(doc) if t.isalpha()]

class LemmaTokenizer:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
```

```
        return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) if t.
 ↪isalpha()]

class LemmaTokenizer_word:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) ]


class StemTokenizer:
     def __init__(self):
       self.wnl =PorterStemmer()
     def __call__(self, doc):
       return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
```

[ ]:
```
# best model
t_start = time.time()

pipe_params = {
    "vect__binary": [False,True],
    "vect__stop_words": [list(stop_words_library)],
    "selecter__k":[5000,3000],
    "normalizer__norm": ['l2','l1','max'],
    'classify__n_estimators':[100]
}

model = RandomForestClassifier()

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
normalizer = Normalizer()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter",␣
 ↪selecter),("normalizer",normalizer),("classify", model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)
```

```
print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
The best accuracy is 89.274.
The winning parameters are {'classify__n_estimators': 100, 'normalizer__norm':
'max', 'selecter__k': 3000, 'vect__binary': True, 'vect__stop_words': ['top',
'them', 'however', 'together', 'sixty', 'such', 'elsewhere', 'done', 'for',
'please', "needn't", 'are', 'and', 'the', 'did', 'during', 'as', 'alone',
'though', 'yourselves', 'same', 'bottom', 'should', 'upon', 'themselves', 'at',
'himself', 'll', "haven't", 'ma', 'hasnt', 've', 'always', 'off', 'beforehand',
'i', 'any', 'whose', 'around', 'your', 'along', 'yours', "you're", 'hereby',
'don', 'he', 'mightn', 'yourself', 'ain', 'often', 'some', 'we', 'mustn',
'thereupon', 'per', 'even', 'since', 'will', 'couldnt', 'that', 'someone',
"you'll", 'anything', 'then', 'etc', 'few', 'other', 'those', 'never', 'once',
'ourselves', 'hadn', 'system', 'take', 'now', 'under', 'who', 'seemed', 'had',
'well', 'whatever', 'weren', 'hers', 'nothing', 'next', "don't", 'aren',
'needn', 'its', 'somewhere', 'up', "wasn't", 'side', 'above', 'third', 'full',
'until', 'not', 'fifteen', 'fill', 'hence', 'or', 'rather', 'five', 'put',
"wouldn't", 'seeming', 'only', 'after', 'theirs', 'one', "shan't", 'o', 'just',
'thru', 's', 'being', 'latter', 'amount', 'whereafter', 'front', 'itself',
'fifty', 'within', "she's", 'these', 'me', 'were', "doesn't", 'against', 'mill',
'whoever', 'thereby', 'wherein', "isn't", 'con', 'twenty', 'anyone', 'least',
'via', 'an', 'nine', "it's", 'seem', 'nevertheless', 'she', 'our', 'am', 'ever',
'thin', 'whence', 'how', 'hasn', 'whereupon', 'amoungst', "hadn't", 'was', 'm',
'him', 'while', 'many', 'too', 'into', 'herself', 'three', 'neither', 'can',
'hereupon', 'but', 'get', "should've", 'inc', 'nobody', 'two', 'on', 'further',
'whom', 'whither', 'their', 'afterwards', 'toward', 'ltd', 'everywhere', 'out',
'due', 'whenever', 'might', 'less', 'mine', 'several', 'of', 'four', 'name',
"weren't", 'they', 'with', 'having', 'hereafter', 'forty', 'eleven', 'shan',
'over', 'have', 'herein', 'go', 'formerly', 'my', 'also', 'another', 'where',
'else', "you've", 'anywhere', 'describe', 'yet', 't', 'none', 'there', 'thence',
'without', 'ie', 'whether', 'besides', 'except', 'y', 'every', 'shouldn', 'why',
'indeed', 'otherwise', "mustn't", 'meanwhile', 'before', 'somehow', 'which',
'so', 'more', 'when', 'cant', 'twelve', 'didn', 'if', 'very', 'see', 'beside',
'mostly', "won't", 'doing', 'from', 'again', 'first', 'nor', 'nowhere',
"aren't", 'seems', 'much', "hasn't", 'becoming', 'find', 'ours', 'a', 'eight',
'thus', 'everything', 'this', 'un', 'isn', 'either', 'give', 'may', 'below',
'move', 'about', 'both', "didn't", 'anyway', 'own', 'cry', 'couldn', 'no', 'eg',
'empty', 'must', 'haven', 'be', 'sometimes', 'ten', 'show', 'here', 'interest',
'what', 'co', 'doesn', 'd', 'between', 'de', 'her', 'made', 'namely', 'won',
'almost', 'hundred', 'across', 'fire', 'latterly', 'cannot', 'whole', 'do',
'among', 'his', 're', 'all', 'became', 'been', 'onto', 'than', 'would',
'moreover', 'becomes', 'although', 'still', "mightn't", 'most', 'to', 'beyond',
'former', 'thick', 'each', 'does', "couldn't", 'sometime', 'found', 'could',
'it', 'us', 'in', 'sincere', 'behind', 'everyone', 'last', "you'd", 'bill',
```

```
'whereas', 'detail', 'through', 'thereafter', 'noone', 'wasn', 'therein', 'six',
'towards', 'anyhow', 'has', 'call', 'myself', 'become', 'therefore', 'whereby',
'keep', 'down', 'by', 'something', 'wouldn', 'already', 'amongst', 'is', 'back',
'you', "shouldn't", 'part', 'enough', 'wherever', 'perhaps', "that'll",
'because', 'others', 'serious', 'throughout']}
Run time: 28.590136528015137 seconds
```

[ ]:
```
y_pred = grid.predict(test_x)
create_test_csv(y_pred,"random_forest_06032023_01.csv")
```

```
File saved.
```

Logistic_Regression

March 12, 2023

[ ]:
```
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from google.colab import drive
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import random
import time
import re
import string
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2,␣
↪f_classif,mutual_info_classif,f_regression
from sklearn.preprocessing import Normalizer
from sklearn import model_selection
from sklearn import svm
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[ ]:
```
True
```

[ ]:
```
#import the data


drive.mount('/content/gdrive/', force_remount=True)

train_data_initial = pd.read_csv('/content/gdrive/MyDrive/train.csv')
test_data = pd.read_csv('/content/gdrive/MyDrive/test.csv')

print('shape train:',train_data_initial.shape)
print('shape test:',test_data.shape)
```

```
Mounted at /content/gdrive/
shape train: (718, 2)
shape test: (279, 2)
```

[ ]:
```
def shuffle_data(df):
    random.seed(0)  # Use a fixed seed for the random number generator
    df = df.sample(frac=1, random_state=0).reset_index(drop=True)
    return df
```

[ ]:
```
#function for creating the test csv file to upload to kaggle
def create_test_csv(data, outfile_name):
  rawdata= {'subreddit':data}
  csv = pd.DataFrame(rawdata, columns = ['subreddit'])
  csv.to_csv(outfile_name,index=True, header=True)
  print ("File saved.")
```

[ ]:
```
#shuffle the data and split the features from the label
train_data = shuffle_data(train_data_initial)



train_x_initial = train_data["body"]
train_y = train_data["subreddit"]
test_x_initial = test_data["body"]
```

[ ]:
```
#remove punctuation from train and test
train_x = train_x_initial.copy()

for i in range(train_x.shape[0]):
  train_x[i]= train_x[i].translate(str.maketrans('', '', string.punctuation))

test_x = test_x_initial.copy()

for i in range(test_x.shape[0]):
  test_x[i]= test_x[i].translate(str.maketrans('', '', string.punctuation))
```

[ ]:
```
print(test_x[5])
#print(test_x_initial[5])
```

```
I like cars with screensas long as the UI is intuitive and phonelike Ive never
driven a new Edge nor have I driven a Ford with Sync 3

As far as I can tell it looks good and concise I like it
```

[ ]:
```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    return text
```

[ ]:
```
def remove_punctuation(text):
   translator = str.maketrans('', '', string.punctuation)
   text = text.translate(translator)
   return text
```

[ ]:
```
#create a dictionary of stop words
stop_words_nltk = set(stopwords.words('english'))
```

```
stop_words_sklearn = text.ENGLISH_STOP_WORDS
stop_words_library = stop_words_sklearn.union(stop_words_nltk)
```

[ ]:
```
#initial training without removing parameters

t_start = time.time()

pipe_params = {
    'classify__penalty': ['l1', 'l2'],   #'classify__penalty': ['l1', 'l2'],
    'classify__C': [0.01, 0.1, 1.0, 10.0],     #'classify__C': [0.01, 0.1, 1.0,␣
 ↪10.0],
    'classify__solver': ['liblinear'],   #  #'classify__solver': ['liblinear',␣
 ↪'newton-cg', 'lbfgs', 'sag', 'saga'],
    'classify__max_iter': [100, 500, 1000],
    'classify__class_weight': [None, 'balanced']
}

vectorizer = CountVectorizer()
model = LogisticRegression()

pipe = Pipeline(
    [("vect", vectorizer),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
The best accuracy is 89.411.
The winning parameters are {'classify__C': 10.0, 'classify__class_weight': None,
'classify__max_iter': 100, 'classify__penalty': 'l2', 'classify__solver':
'liblinear'}
Run time: 32.65085744857788 seconds
```

[ ]:
```
#initial training with stop words  93.037

t_start = time.time()
```

```
pipe_params = {
    'classify__penalty': ['l2'],   #'classify__penalty': ['l1', 'l2'],
    'classify__C': [10.0],     #'classify__C': [0.01, 0.1, 1.0, 10.0],
    'classify__solver': ['sag'],   #'classify__solver': ['liblinear',␣
 ↪'newton-cg', 'lbfgs', 'sag', 'saga'],
    'classify__max_iter': [1000], # 'classify__max_iter': [100, 500, 1000],
    #'classify__class_weight': ['balanced'],     #'classify__class_weight':␣
 ↪[None, 'balanced'],
    "vect__stop_words": [list(stop_words_nltk), list(stop_words_sklearn),␣
 ↪list(stop_words_library)],
    "selecter__k":[5000]
}




#stop_words_nltk
#stop_words_sklearn


vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = LogisticRegression()



pipe = Pipeline(
     [("vect", vectorizer),("selecter", selecter),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
The best accuracy is 93.037.
The winning parameters are {'classify__C': 10.0, 'classify__max_iter': 1000,
```

'classify__penalty': 'l2', 'classify__solver': 'sag', 'selecter__k': 5000,
'vect__stop_words': ['thereupon', 'latter', 'up', 'aren', 'almost', "that'll",
'among', 'their', 'both', 'never', 'been', 'further', 'along', 'whoever', 'inc',
"you'd", 'again', 'eleven', 'another', 'his', 'empty', 'always', 'though',
'within', 'bill', 'put', 'needn', 'behind', 'could', 'all', 'four',
'beforehand', 'otherwise', 'well', 'upon', 'often', "won't", 'under', 'its',
'perhaps', 'fire', 'un', "don't", 'already', 'from', 'six', 'still', 'get',
'go', 'as', 'during', 'toward', 'sixty', 'not', 'isn', 'amongst', 'had',
"didn't", 'see', 'first', 'nowhere', "wouldn't", 'mostly', 'thin', 'detail',
'hence', 'ours', 'against', 'being', 'seem', 'they', 'keep', 'none', 'less',
'become', 'where', 'sometimes', 'move', 'mill', 'nine', 'hereafter', 'yours',
'few', 'i', 'much', 'twenty', 'has', 'last', 'or', 'while', 'anything', 'don',
'top', 'whereas', 'something', 'hasn', 'such', 'between', 'down', 'm', 'off',
'even', 'others', 'we', 'system', 'after', 'therein', 'thereafter', 'this',
'weren', 'cant', 'two', 'which', 'thus', 'wouldn', 'fifteen', 'can', 'wasn',
'will', 'made', 'are', 'once', 'several', 'third', 'whenever', 'must',
'themselves', "shan't", 'front', 'about', 'a', 'amoungst', 'some', "hadn't",
'show', 'whether', 'formerly', 'sincere', 'the', 'with', 'more', 'side',
'there', "you're", 'mine', 'however', 'whereafter', 'give', 'ain', 'mustn',
'here', 'became', 'did', 'now', "isn't", 'shan', 'he', "hasn't", 'any', 'ma',
'nobody', 'didn', 'theirs', 'until', 'you', 'it', 'hereupon', 'above', 'noone',
'least', 'becomes', 'hasnt', 'other', 'them', 'were', 'someone', 'eg', "she's",
'con', 'take', 'haven', 'serious', 'when', 'alone', 'my', 'anyone', "aren't",
'fifty', 'was', 'please', "should've", 'nevertheless', 'de', 'doesn', 'since',
"shouldn't", 'latterly', 'although', 'name', 'cry', 'ourselves', 'too', 'is',
'and', 'herein', 'an', 'full', 'y', 'very', 'forty', 'itself', 'me', 'before',
'd', 'do', 'moreover', 'back', 'eight', 'most', 'so', 'therefore', 'via', 'who',
'thereby', 'through', 'shouldn', 'co', 'many', 'hadn', 'whom', "it's",
'yourselves', 'ltd', 'due', 'somewhere', 's', 'ie', 'done', 'afterwards',
'himself', 'onto', 'call', 'beyond', 'below', 'yourself', 'everyone', "haven't",
'him', 'just', 'seeming', 'does', 'may', 'per', 'find', 'because', 'if',
'would', 'whence', 'cannot', 'enough', 'on', 'five', 'ever', 'across',
'herself', 'to', "couldn't", 'us', "doesn't", 'wherein', 'thence', 'in',
'describe', 'whole', 'whatever', 'everything', 'elsewhere', "you've", "mustn't",
'fill', 'your', 're', 'might', 'twelve', 'having', 'besides', 'am', 'nor', 've',
"needn't", 'else', 'hers', 'what', 'hundred', 'beside', 'thick', 'o', 'either',
'throughout', 'only', 'anywhere', 'couldn', "weren't", 'those', 'interest',
'amount', 'neither', 'each', 'doing', 'bottom', 'sometime', 'next', 'without',
'meanwhile', 'seemed', 'except', 'why', 'for', 'over', "you'll", 'whereby',
'same', "wasn't", 'former', 'wherever', 't', 'around', 'mightn', 'one', 'she',
'no', 'part', 'anyhow', 'couldnt', 'own', 'three', 'rather', 'won', 'our',
"mightn't", 'that', 'yet', 'etc', 'by', 'indeed', 'into', 'at', 'also', 'thru',
'how', 'than', 'towards', 'ten', 'myself', 'of', 'then', 'll', 'anyway',
'hereby', 'her', 'should', 'whither', 'be', 'seems', 'have', 'nothing',
'everywhere', 'whereupon', 'these', 'together', 'becoming', 'but', 'out',
'every', 'namely', 'somehow', 'found', 'whose']}
Run time: 11.839037895202637 seconds

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

[ ]:
```
#stem lemmatizer
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

class LemmaTokenizer_Pos:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in␣
 ↪word_tokenize(doc) if t.isalpha()]

class LemmaTokenizer:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) if t.
 ↪isalpha()]

class LemmaTokenizer_word:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) ]


class StemTokenizer:
     def __init__(self):
       self.wnl =PorterStemmer()
     def __call__(self, doc):
       return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
```

[ ]:
```
stop_words_custom = [
# All pronouns and associated words
"i","i'll","i'd","i'm","i've","ive","me","myself","you","you'll","you'd","you're","you've","yo
"he'd",
"he's",
```

```
"him",
"she",
"she'll",
"she'd",
"she's",
"her",
"it",
"it'll",
"it'd",
"it's",
"itself",
"oneself",
"we",
"we'll",
"we'd",
"we're",
"we've",
"us",
"ourselves",
"they",
"they'll",
"they'd",
"they're",
"they've",
"them",
"themselves",
"everyone",
"everyone's",
"everybody",
"everybody's",
"someone",
"someone's",
"somebody",
"somebody's",
"nobody",
"nobody's",
"anyone",
"anyone's",
"everything",
"everything's",
"something",
"something's",
"nothing",
"nothing's",
"anything",
"anything's",
# All determiners and associated words
```

```
"a",
"an",
"the",
"this",
"that",
"that's",
"these",
"those",
"my",
#"mine",   #Omitted since mine can refer to something else
"your",
"yours",
"his",
"hers",
"its",
"our",
"ours",
"own",
"their",
"theirs",
"few",
"much",
"many",
"lot",
"lots",
"some",
"any",
"enough",
"all",
"both",
"half",
"either",
"neither",
"each",
"every",
"certain",
"other",
"another",
"such",
"several",
"multiple",
# "what",#Dealt with later on
"rather",
"quite",
# All prepositions
"aboard",
"about",
```

```
"above",
"across",
"after",
"against",
"along",
"amid",
"amidst",
"among",
"amongst",
"anti",
"around",
"as",
"at",
"away",
"before",
"behind",
"below",
"beneath",
"beside",
"besides",
"between",
"beyond",
"but",
"by",
"concerning",
"considering",
"despite",
"down",
"during",
"except",
"excepting",
"excluding",
"far",
"following",
"for",
"from",
"here",
"here's",
"in",
"inside",
"into",
"left",
"like",
"minus",
"near",
"of",
"off",
```

```
"on",
"onto",
"opposite",
"out",
"outside",
"over",
"past",
"per",
"plus",
"regarding",
"right",
#"round",   #Omitted
#"save",#Omitted
"since",
"than",
"there",
"there's",
"through",
"to",
"toward",
"towards",
"under",
"underneath",
"unlike",
"until",
"up",
"upon",
"versus",
"via",
"with",
"within",
"without",
# Irrelevant verbs
"may",
"might",
"will",
"won't",
"would",
"wouldn't",
"can",
"can't",
"cannot",
"could",
"couldn't",
"should",
"shouldn't",
"must",
```

```
"must've",
"be",
"being",
"been",
"am",
"are",
"aren't",
"ain't",
"is",
"isn't",
"was",
"wasn't",
"were",
"weren't",
"do",
"doing",
"don't",
"does",
"doesn't",
"did",
"didn't",
"done",
"have",
"haven't",
"having",
"has",
"hasn't",
"had",
"hadn't",
"get",
"getting",
"gets",
"got",
"gotten",
"go",
"going",
"gonna",
"goes",
"went",
"gone",
"make",
"making",
"makes",
"made",
"take",
"taking",
"takes",
```

```
"took",
"taken",
"need",
"needing",
"needs",
"needed",
"use",
"using",
"uses",
"used",
"want",
"wanna",
"wanting",
"wants",
"let",
"lets",
"letting",
"let's",
"suppose",
"supposing",
"supposes",
"supposed",
"seem",
"seeming",
"seems",
"seemed",
"say",
"saying",
"says",
"said",
"know",
"knowing",
"knows",
"knew",
"known",
"look",
"looking",
"looked",
"think",
"thinking",
"thinks",
"thought",
"feel",
"feels",
"felt",
"based",
"put",
```

```
"puts",
#"wanted"   #Omitted since the advective is relevant
# Question words and associated words
"who",
"who's",
"who've",
"who'd",
"whoever",
"whoever's",
"whom",
"whomever",
"whomever's",
"whose",
"whosever",
"whosever's",
"when",
"whenever",
"which",
"whichever",
"where",
"where's",
"where'd",
"wherever",
"why",
"why's",
"why'd",
"whyever",
"what",
"what's",
"whatever",
"whence",
"how",
"how's",
"how'd",
"however",
"whether",
"whatsoever",
# Connector words and irrelevant adverbs
"and",
"or",
"not",
"because",
"also",
"always",
"never",
"only",
"really",
```

```
"very",
"greatly",
"extremely",
"somewhat",
"no",
"nope",
"nah",
"yes",
"yep",
"yeh",
"yeah",
"maybe",
"perhaps",
"more",
"most",
"less",
"least",
"good",
"great",
"well",
"better",
"best",
"bad",
"worse",
"worst",
"too",
"thru",
"though",
"although",
"yet",
"already",
"then",
"even",
"now",
"sometimes",
"still",
"together",
"altogether",
"entirely",
"fully",
"entire",
"whole",
"completely",
"utterly",
"seemingly",
"apparently",
"clearly",
```

```
"obviously",
"actually",
"actual",
"usually",
"usual",
"literally",
"honestly",
"absolutely",
"definitely",
"generally",
"totally",
"finally",
"basically",
"essentially",
"fundamentally",
"automatically",
"immediately",
"necessarily",
"primarily",
"normally",
"perfectly",
"constantly",
"particularly",
"eventually",
"hopefully",
"mainly",
"typically",
"specifically",
"differently",
"appropriately",
"plenty",
"certainly",
"unfortunately",
"ultimately",
"unlikely",
"likely",
"potentially",
"fortunately",
"personally",
"directly",
"indirectly",
"nearly",
"closely",
"slightly",
"probably",
"possibly",
"especially",
```

```
"frequently",
"often",
"oftentimes",
"seldom",
"rarely",
"sure",
"while",
"whilst",
"able",
"unable",
"else",
"ever",
"once",
"twice",
"thrice",
"almost",
"again",
"instead",
"next",
"previous",
"unless",
"somehow",
"anyhow",
"anywhere",
"somewhere",
"everywhere",
"nowhere",
"further",
"anymore",
"later",
"ago",
"ahead",
"just",
"same",
"different",
"big",
"small",
"little",
"tiny",
"large",
"huge",
"pretty",
"mostly",
"anyway",
"anyways",
"otherwise",
"regardless",
```

```
"throughout",
"additionally",
"moreover",
"furthermore",
"meanwhile",
"afterwards",
# Irrelevant nouns
"thing",
"thing's",
"things",
"stuff",
"other's",
"others",
"another's",
"total",
"",
"false",
"none",
"way",
"kind",
# Lettered numbers and order
"zero",
"zeros",
"zeroes",
"one",
"ones",
"two",
"three",
"four",
"five",
"six",
"seven",
"eight",
"nine",
"ten",
"twenty",
"thirty",
"forty",
"fifty",
"sixty",
"seventy",
"eighty",
"ninety",
"hundred",
"hundreds",
"thousand",
"thousands",
```

```
"million",
"millions",
"first",
"last",
"second",
"third",
"fourth",
"fifth",
"sixth",
"seventh",
"eigth",
"ninth",
"tenth",
"firstly",
"secondly",
"thirdly",
"lastly",
# Greetings and slang
"hello",
"hi",
"hey",
"sup",
"yo",
"greetings",
"please",
"okay",
"ok",
"y'all",
"lol",
"rofl",
"thank",
"thanks",
"alright",
"kinda",
"dont",
"sorry",
"idk",
"tldr",
"tl",
"dr",  #This means that dr (doctor) is a bad feature because of tl;dr
"tbh",
"dude",
"tho",
"aka",
"plz",
"pls",
"bit",
```

```
"don",
# Miscellaneous
"www",
"https",
"http",
"com",
"etc"
"html",
"reddit",
"subreddit",
"subreddits",
"comments",
"reply",
"replies",
"thread",
"threads",
"post",
"posts",
"website",
"websites",
"web site",
"web sites"]
print('length custom:',len(stop_words_custom))
```

```
length custom: 589
```

[ ]:
```
print(len(stop_words_custom))
```

```
589
```

[ ]:
```
#function for creating the test csv file to upload to kaggle
def create_test_csv(data, outfile_name):
  rawdata= {'subreddit':data}
  csv = pd.DataFrame(rawdata, columns = ['subreddit'])
  csv.to_csv(outfile_name,index=True, header=True)
  print ("File saved.")
```

[ ]:
```
#initial training with stop words. LemmaTokenizer_word

t_start = time.time()

pipe_params = {
    'classify__penalty': ['l2'],   #'classify__penalty': ['l1', 'l2'],
    'classify__C': [10.0],     #'classify__C': [0.01, 0.1, 1.0, 10.0],
    'classify__solver': ['sag'],   #'classify__solver': ['liblinear',␣
 ↪'newton-cg', 'lbfgs', 'sag', 'saga'],
    'classify__max_iter': [1000], # 'classify__max_iter': [100, 500, 1000],
```

```
    #'classify__class_weight': [None],     #'classify__class_weight': [None,␣
 ↪'balanced'],
    "vect__stop_words": [list(stop_words_nltk), list(stop_words_sklearn),␣
 ↪list(stop_words_library)],
    "selecter__k":[5000],
    #"vect__tokenizer": [LemmaTokenizer_word()]
}




#stop_words_nltk
#stop_words_sklearn


vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = LogisticRegression()



pipe = Pipeline(
     [("vect", vectorizer),("selecter", selecter),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
The best accuracy is 93.037.
The winning parameters are {'classify__C': 10.0, 'classify__max_iter': 1000,
'classify__penalty': 'l2', 'classify__solver': 'sag', 'selecter__k': 5000,
'vect__stop_words': ['most', 'through', 'everything', 'had', 'have', 'these',
'did', 'un', 'still', 'anyone', 'her', 'almost', 'mine', "hadn't", 'its', 'one',
"shouldn't", 'thence', 'never', 'your', 'doing', 'out', 'three', 'some', 'due',
'below', 'although', 'wasn', 'made', 'very', 'other', 'what', 'bill', 'am',
'as', 'see', 'cant', 'whose', 'fifty', 'wherein', 'amount', 'twenty', 'nobody',
```

```
'somewhere', "you're", 'hereafter', 'along', 've', 'hence', 'against', 'hadn',
'often', 'noone', 'more', 'fifteen', 'becomes', 'seem', 'mustn', 'ltd', 'upon',
'two', "haven't", 'won', 'among', 'something', "aren't", 'them', 'do', 'then',
'yourselves', 'give', 'onto', "needn't", 'whither', 'under', 'last', "mightn't",
'seems', 'shan', "won't", 'becoming', 'therefore', 'after', 'done', 'i',
'couldnt', 'another', 'put', 'towards', 'myself', "you'd", 'yet', "shan't",
'all', 'be', 'back', 'hers', 'you', 'from', 'on', "wouldn't", 'wherever', 'not',
'y', 'if', 'because', 'become', 'such', 'so', 'an', 'co', 'once', 'move',
'several', 'ourselves', 'even', 'nowhere', 'ours', 'himself', 'toward', 're',
'hasn', 'whence', 'him', 'must', 'meanwhile', 'there', 'four', 'behind',
"doesn't", 'ain', 'whereupon', 'needn', 'anything', 'where', 'together', 'well',
'everyone', 'else', 'none', 'don', 'couldn', 'take', 'should', 'than', 'anyhow',
'might', 'further', 'whatever', 'someone', 'mightn', 'who', 'thereupon',
'across', 'full', 'least', 'throughout', 'twelve', 'haven', 'being', 'namely',
'call', 'isn', 'ever', 'until', 'yours', 'will', 'inc', "hasn't", 'm',
"weren't", 'whoever', 'my', 'down', 'at', 'sometime', 'she', 't', 'herein',
'itself', 'part', 'sixty', 'here', "couldn't", 'he', 'theirs', 'whereas',
'otherwise', 'yourself', 'that', 'again', 'forty', 's', 'always', 'which',
'bottom', 'how', 'can', 'go', 'hereupon', 'since', 'just', 'latterly', 'could',
'hereby', 'll', 'mostly', "you've", 'much', 'seemed', "mustn't", 'was', 'our',
'without', 'beforehand', 'serious', 'via', 'me', 'formerly', 'why', 'enough',
"should've", 'whereafter', 'perhaps', 'sincere', 'five', 'many', 'now',
'thereafter', 'about', 'detail', 'and', 'wouldn', 'cannot', 'having', "didn't",
'it', 'eleven', 'nor', 'cry', 'either', 'thin', 'sometimes', 'seeming', 'we',
'd', 'con', 'same', 'to', 'per', 'his', 'the', 'fire', 'found', 'describe',
'already', 'within', 'whether', 'doesn', 'latter', 'has', 'therein', 'rather',
'of', 'anywhere', 'amongst', 'ten', 'o', 'would', 'front', 'de', 'alone',
'system', 'elsewhere', 'those', 'for', 'thick', 'etc', 'a', 'are', 'find',
'though', 'neither', 'whereby', 'own', 'over', 'only', 'thereby', "don't", 'no',
'whenever', 'themselves', 'also', 'beside', 'nothing', 'thus', 'ie', 'third',
'aren', 'too', 'during', 'off', 'became', 'didn', 'fill', 'indeed', 'please',
'in', 'hasnt', 'hundred', 'afterwards', 'mill', 'name', 'their', 'former',
'but', 'moreover', 'thru', 'however', 'whole', 'been', 'next', 'besides', 'eg',
'side', "you'll", 'first', 'keep', 'somehow', 'weren', 'each', 'nevertheless',
'up', 'is', 'they', 'amoungst', 'any', 'everywhere', 'around', 'empty', "isn't",
'anyway', 'shouldn', "that'll", 'nine', 'beyond', 'while', 'whom', 'were',
'top', "she's", 'interest', 'show', 'get', 'ma', 'less', 'between', 'by',
'does', 'herself', 'few', 'above', 'into', 'with', 'six', 'may', 'except',
'eight', "wasn't", 'others', "it's", 'us', 'both', 'every', 'this', 'when',
'or', 'before']}
Run time: 15.71203327178955 seconds

/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

[ ]:
```
#initial training with stop words

t_start = time.time()

pipe_params = {
    'classify__penalty': ['l2'],   #'classify__penalty': ['l1', 'l2'],
    'classify__C': [10.0],     #'classify__C': [0.01, 0.1, 1.0, 10.0],
    'classify__solver': ['sag'],   #'classify__solver': ['liblinear',␣
 ↪'newton-cg', 'lbfgs', 'sag', 'saga'],
    'classify__max_iter': [1000], # 'classify__max_iter': [100, 500, 1000],
    #'classify__class_weight': ['balanced'],     #'classify__class_weight':␣
 ↪[None, 'balanced'],
    "vect__stop_words": [list(stop_words_nltk), list(stop_words_sklearn),␣
 ↪list(stop_words_library)],
    "selecter__k":[5000,3000],
    "vect__ngram_range":[(1,1)]
}



vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = LogisticRegression()



pipe = Pipeline(
     [("vect", vectorizer),("selecter",␣
 ↪selecter),("normalizer",normalizer),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
The best accuracy is 92.339.
The winning parameters are {'classify__C': 10.0, 'classify__max_iter': 1000,
```

'classify__penalty': 'l2', 'classify__solver': 'sag', 'selecter__k': 5000,
'vect__ngram_range': (1, 1), 'vect__stop_words': ['most', 'through',
'everything', 'had', 'have', 'these', 'did', 'un', 'still', 'anyone', 'her',
'almost', 'mine', "hadn't", 'its', 'one', "shouldn't", 'thence', 'never',
'your', 'doing', 'out', 'three', 'some', 'due', 'below', 'although', 'wasn',
'made', 'very', 'other', 'what', 'bill', 'am', 'as', 'see', 'cant', 'whose',
'fifty', 'wherein', 'amount', 'twenty', 'nobody', 'somewhere', "you're",
'hereafter', 'along', 've', 'hence', 'against', 'hadn', 'often', 'noone',
'more', 'fifteen', 'becomes', 'seem', 'mustn', 'ltd', 'upon', 'two', "haven't",
'won', 'among', 'something', "aren't", 'them', 'do', 'then', 'yourselves',
'give', 'onto', "needn't", 'whither', 'under', 'last', "mightn't", 'seems',
'shan', "won't", 'becoming', 'therefore', 'after', 'done', 'i', 'couldnt',
'another', 'put', 'towards', 'myself', "you'd", 'yet', "shan't", 'all', 'be',
'back', 'hers', 'you', 'from', 'on', "wouldn't", 'wherever', 'not', 'y', 'if',
'because', 'become', 'such', 'so', 'an', 'co', 'once', 'move', 'several',
'ourselves', 'even', 'nowhere', 'ours', 'himself', 'toward', 're', 'hasn',
'whence', 'him', 'must', 'meanwhile', 'there', 'four', 'behind', "doesn't",
'ain', 'whereupon', 'needn', 'anything', 'where', 'together', 'well',
'everyone', 'else', 'none', 'don', 'couldn', 'take', 'should', 'than', 'anyhow',
'might', 'further', 'whatever', 'someone', 'mightn', 'who', 'thereupon',
'across', 'full', 'least', 'throughout', 'twelve', 'haven', 'being', 'namely',
'call', 'isn', 'ever', 'until', 'yours', 'will', 'inc', "hasn't", 'm',
"weren't", 'whoever', 'my', 'down', 'at', 'sometime', 'she', 't', 'herein',
'itself', 'part', 'sixty', 'here', "couldn't", 'he', 'theirs', 'whereas',
'otherwise', 'yourself', 'that', 'again', 'forty', 's', 'always', 'which',
'bottom', 'how', 'can', 'go', 'hereupon', 'since', 'just', 'latterly', 'could',
'hereby', 'll', 'mostly', "you've", 'much', 'seemed', "mustn't", 'was', 'our',
'without', 'beforehand', 'serious', 'via', 'me', 'formerly', 'why', 'enough',
"should've", 'whereafter', 'perhaps', 'sincere', 'five', 'many', 'now',
'thereafter', 'about', 'detail', 'and', 'wouldn', 'cannot', 'having', "didn't",
'it', 'eleven', 'nor', 'cry', 'either', 'thin', 'sometimes', 'seeming', 'we',
'd', 'con', 'same', 'to', 'per', 'his', 'the', 'fire', 'found', 'describe',
'already', 'within', 'whether', 'doesn', 'latter', 'has', 'therein', 'rather',
'of', 'anywhere', 'amongst', 'ten', 'o', 'would', 'front', 'de', 'alone',
'system', 'elsewhere', 'those', 'for', 'thick', 'etc', 'a', 'are', 'find',
'though', 'neither', 'whereby', 'own', 'over', 'only', 'thereby', "don't", 'no',
'whenever', 'themselves', 'also', 'beside', 'nothing', 'thus', 'ie', 'third',
'aren', 'too', 'during', 'off', 'became', 'didn', 'fill', 'indeed', 'please',
'in', 'hasnt', 'hundred', 'afterwards', 'mill', 'name', 'their', 'former',
'but', 'moreover', 'thru', 'however', 'whole', 'been', 'next', 'besides', 'eg',
'side', "you'll", 'first', 'keep', 'somehow', 'weren', 'each', 'nevertheless',
'up', 'is', 'they', 'amoungst', 'any', 'everywhere', 'around', 'empty', "isn't",
'anyway', 'shouldn', "that'll", 'nine', 'beyond', 'while', 'whom', 'were',
'top', "she's", 'interest', 'show', 'get', 'ma', 'less', 'between', 'by',
'does', 'herself', 'few', 'above', 'into', 'with', 'six', 'may', 'except',
'eight', "wasn't", 'others', "it's", 'us', 'both', 'every', 'this', 'when',
'or', 'before']}
Run time: 9.341354131698608 seconds

```
[ ]: #initial training with stop words. 93.038

t_start = time.time()

pipe_params = {
    'classify__penalty': ['l2'],   #'classify__penalty': ['l1', 'l2'],
    'classify__C': [10.0],     #'classify__C': [0.01, 0.1, 1.0, 10.0],
    'classify__solver': ['sag'],   #'classify__solver': ['liblinear',␣
 ↪'newton-cg', 'lbfgs', 'sag', 'saga'],
    'classify__max_iter': [1000], # 'classify__max_iter': [100, 500, 1000],
    'classify__class_weight': [None, 'balanced'],     #'classify__class_weight':
 ↪ [None, 'balanced'],
    "vect__stop_words": [list(stop_words_library)],  ##[list(stop_words_nltk),␣
 ↪list(stop_words_sklearn), list(stop_words_library)]
    "selecter__k":[5000],
    "vect__ngram_range":[(1,1)]

}




#stop_words_nltk
#stop_words_sklearn


vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = LogisticRegression()



pipe = Pipeline(
     [("vect", vectorizer),("selecter", selecter),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
```

```
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
The best accuracy is 92.899.
The winning parameters are {'classify__C': 10.0, 'classify__class_weight': None,
'classify__max_iter': 1000, 'classify__penalty': 'l2', 'classify__solver':
'sag', 'selecter__k': 5000, 'vect__ngram_range': (1, 1), 'vect__stop_words':
['most', 'through', 'everything', 'had', 'have', 'these', 'did', 'un', 'still',
'anyone', 'her', 'almost', 'mine', "hadn't", 'its', 'one', "shouldn't",
'thence', 'never', 'your', 'doing', 'out', 'three', 'some', 'due', 'below',
'although', 'wasn', 'made', 'very', 'other', 'what', 'bill', 'am', 'as', 'see',
'cant', 'whose', 'fifty', 'wherein', 'amount', 'twenty', 'nobody', 'somewhere',
"you're", 'hereafter', 'along', 've', 'hence', 'against', 'hadn', 'often',
'noone', 'more', 'fifteen', 'becomes', 'seem', 'mustn', 'ltd', 'upon', 'two',
"haven't", 'won', 'among', 'something', "aren't", 'them', 'do', 'then',
'yourselves', 'give', 'onto', "needn't", 'whither', 'under', 'last', "mightn't",
'seems', 'shan', "won't", 'becoming', 'therefore', 'after', 'done', 'i',
'couldnt', 'another', 'put', 'towards', 'myself', "you'd", 'yet', "shan't",
'all', 'be', 'back', 'hers', 'you', 'from', 'on', "wouldn't", 'wherever', 'not',
'y', 'if', 'because', 'become', 'such', 'so', 'an', 'co', 'once', 'move',
'several', 'ourselves', 'even', 'nowhere', 'ours', 'himself', 'toward', 're',
'hasn', 'whence', 'him', 'must', 'meanwhile', 'there', 'four', 'behind',
"doesn't", 'ain', 'whereupon', 'needn', 'anything', 'where', 'together', 'well',
'everyone', 'else', 'none', 'don', 'couldn', 'take', 'should', 'than', 'anyhow',
'might', 'further', 'whatever', 'someone', 'mightn', 'who', 'thereupon',
'across', 'full', 'least', 'throughout', 'twelve', 'haven', 'being', 'namely',
'call', 'isn', 'ever', 'until', 'yours', 'will', 'inc', "hasn't", 'm',
"weren't", 'whoever', 'my', 'down', 'at', 'sometime', 'she', 't', 'herein',
'itself', 'part', 'sixty', 'here', "couldn't", 'he', 'theirs', 'whereas',
'otherwise', 'yourself', 'that', 'again', 'forty', 's', 'always', 'which',
'bottom', 'how', 'can', 'go', 'hereupon', 'since', 'just', 'latterly', 'could',
'hereby', 'll', 'mostly', "you've", 'much', 'seemed', "mustn't", 'was', 'our',
'without', 'beforehand', 'serious', 'via', 'me', 'formerly', 'why', 'enough',
"should've", 'whereafter', 'perhaps', 'sincere', 'five', 'many', 'now',
'thereafter', 'about', 'detail', 'and', 'wouldn', 'cannot', 'having', "didn't",
'it', 'eleven', 'nor', 'cry', 'either', 'thin', 'sometimes', 'seeming', 'we',
'd', 'con', 'same', 'to', 'per', 'his', 'the', 'fire', 'found', 'describe',
'already', 'within', 'whether', 'doesn', 'latter', 'has', 'therein', 'rather',
'of', 'anywhere', 'amongst', 'ten', 'o', 'would', 'front', 'de', 'alone',
'system', 'elsewhere', 'those', 'for', 'thick', 'etc', 'a', 'are', 'find',
'though', 'neither', 'whereby', 'own', 'over', 'only', 'thereby', "don't", 'no',
'whenever', 'themselves', 'also', 'beside', 'nothing', 'thus', 'ie', 'third',
'aren', 'too', 'during', 'off', 'became', 'didn', 'fill', 'indeed', 'please',
'in', 'hasnt', 'hundred', 'afterwards', 'mill', 'name', 'their', 'former',
'but', 'moreover', 'thru', 'however', 'whole', 'been', 'next', 'besides', 'eg',
'side', "you'll", 'first', 'keep', 'somehow', 'weren', 'each', 'nevertheless',
'up', 'is', 'they', 'amoungst', 'any', 'everywhere', 'around', 'empty', "isn't",
```

```
'anyway', 'shouldn', "that'll", 'nine', 'beyond', 'while', 'whom', 'were',
'top', "she's", 'interest', 'show', 'get', 'ma', 'less', 'between', 'by',
'does', 'herself', 'few', 'above', 'into', 'with', 'six', 'may', 'except',
'eight', "wasn't", 'others', "it's", 'us', 'both', 'every', 'this', 'when',
'or', 'before']}
Run time: 9.679741621017456 seconds
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

[ ]:
```
#initial training with stop words.

t_start = time.time()

pipe_params = {
    'classify__penalty': ['l2'],   #'classify__penalty': ['l1', 'l2'],
    'classify__C': [10.0],     #'classify__C': [0.01, 0.1, 1.0, 10.0],
    'classify__solver': ['sag'],   #'classify__solver': ['liblinear',␣
 ↪'newton-cg', 'lbfgs', 'sag', 'saga'],
    'classify__max_iter': [1000], # 'classify__max_iter': [100, 500, 1000],
    'classify__class_weight': [None, 'balanced'],     #'classify__class_weight':
 ↪ [None, 'balanced'],
    "vect__stop_words": [list(stop_words_nltk), list(stop_words_sklearn),␣
 ↪list(stop_words_library)],  ##[list(stop_words_nltk),␣
 ↪list(stop_words_sklearn), list(stop_words_library)]
    "selecter__k":[5000],
    "vect__ngram_range":[(1,1)],
    # "vect__binary": [False]
    #"vect__preprocessor": [preprocess_text,remove_punctuation,None]
    #"vect__binary": [False]

}



vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = LogisticRegression()
#normalizer = Normalizer()


pipe = Pipeline(
     [("vect", vectorizer),("selecter", selecter),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)
```

```
grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
The best accuracy is 93.038.
The winning parameters are {'classify__C': 10.0, 'classify__class_weight': None,
'classify__max_iter': 1000, 'classify__penalty': 'l2', 'classify__solver':
'sag', 'selecter__k': 5000, 'vect__ngram_range': (1, 1), 'vect__stop_words':
['most', 'through', 'everything', 'had', 'have', 'these', 'did', 'un', 'still',
'anyone', 'her', 'almost', 'mine', "hadn't", 'its', 'one', "shouldn't",
'thence', 'never', 'your', 'doing', 'out', 'three', 'some', 'due', 'below',
'although', 'wasn', 'made', 'very', 'other', 'what', 'bill', 'am', 'as', 'see',
'cant', 'whose', 'fifty', 'wherein', 'amount', 'twenty', 'nobody', 'somewhere',
"you're", 'hereafter', 'along', 've', 'hence', 'against', 'hadn', 'often',
'noone', 'more', 'fifteen', 'becomes', 'seem', 'mustn', 'ltd', 'upon', 'two',
"haven't", 'won', 'among', 'something', "aren't", 'them', 'do', 'then',
'yourselves', 'give', 'onto', "needn't", 'whither', 'under', 'last', "mightn't",
'seems', 'shan', "won't", 'becoming', 'therefore', 'after', 'done', 'i',
'couldnt', 'another', 'put', 'towards', 'myself', "you'd", 'yet', "shan't",
'all', 'be', 'back', 'hers', 'you', 'from', 'on', "wouldn't", 'wherever', 'not',
'y', 'if', 'because', 'become', 'such', 'so', 'an', 'co', 'once', 'move',
'several', 'ourselves', 'even', 'nowhere', 'ours', 'himself', 'toward', 're',
'hasn', 'whence', 'him', 'must', 'meanwhile', 'there', 'four', 'behind',
"doesn't", 'ain', 'whereupon', 'needn', 'anything', 'where', 'together', 'well',
'everyone', 'else', 'none', 'don', 'couldn', 'take', 'should', 'than', 'anyhow',
'might', 'further', 'whatever', 'someone', 'mightn', 'who', 'thereupon',
'across', 'full', 'least', 'throughout', 'twelve', 'haven', 'being', 'namely',
'call', 'isn', 'ever', 'until', 'yours', 'will', 'inc', "hasn't", 'm',
"weren't", 'whoever', 'my', 'down', 'at', 'sometime', 'she', 't', 'herein',
'itself', 'part', 'sixty', 'here', "couldn't", 'he', 'theirs', 'whereas',
'otherwise', 'yourself', 'that', 'again', 'forty', 's', 'always', 'which',
'bottom', 'how', 'can', 'go', 'hereupon', 'since', 'just', 'latterly', 'could',
'hereby', 'll', 'mostly', "you've", 'much', 'seemed', "mustn't", 'was', 'our',
'without', 'beforehand', 'serious', 'via', 'me', 'formerly', 'why', 'enough',
"should've", 'whereafter', 'perhaps', 'sincere', 'five', 'many', 'now',
'thereafter', 'about', 'detail', 'and', 'wouldn', 'cannot', 'having', "didn't",
'it', 'eleven', 'nor', 'cry', 'either', 'thin', 'sometimes', 'seeming', 'we',
'd', 'con', 'same', 'to', 'per', 'his', 'the', 'fire', 'found', 'describe',
```

```
'already', 'within', 'whether', 'doesn', 'latter', 'has', 'therein', 'rather',
'of', 'anywhere', 'amongst', 'ten', 'o', 'would', 'front', 'de', 'alone',
'system', 'elsewhere', 'those', 'for', 'thick', 'etc', 'a', 'are', 'find',
'though', 'neither', 'whereby', 'own', 'over', 'only', 'thereby', "don't", 'no',
'whenever', 'themselves', 'also', 'beside', 'nothing', 'thus', 'ie', 'third',
'aren', 'too', 'during', 'off', 'became', 'didn', 'fill', 'indeed', 'please',
'in', 'hasnt', 'hundred', 'afterwards', 'mill', 'name', 'their', 'former',
'but', 'moreover', 'thru', 'however', 'whole', 'been', 'next', 'besides', 'eg',
'side', "you'll", 'first', 'keep', 'somehow', 'weren', 'each', 'nevertheless',
'up', 'is', 'they', 'amoungst', 'any', 'everywhere', 'around', 'empty', "isn't",
'anyway', 'shouldn', "that'll", 'nine', 'beyond', 'while', 'whom', 'were',
'top', "she's", 'interest', 'show', 'get', 'ma', 'less', 'between', 'by',
'does', 'herself', 'few', 'above', 'into', 'with', 'six', 'may', 'except',
'eight', "wasn't", 'others', "it's", 'us', 'both', 'every', 'this', 'when',
'or', 'before']}
Run time: 30.26263689994812 seconds
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

[ ]:
```python
#initial training with stop words. 93.038

t_start = time.time()

pipe_params = {
    'classify__penalty': ['l2'],   #'classify__penalty': ['l1', 'l2'],
    'classify__C': [10.0],     #'classify__C': [0.01, 0.1, 1.0, 10.0],
    'classify__solver': ['sag'],   #'classify__solver': ['liblinear',␣
 ↪'newton-cg', 'lbfgs', 'sag', 'saga'],
    'classify__max_iter': [1000], # 'classify__max_iter': [100, 500, 1000],
    'classify__class_weight': [None, 'balanced'],     #'classify__class_weight':
 ↪ [None, 'balanced'],
    "vect__stop_words": [list(stop_words_nltk), list(stop_words_sklearn),␣
 ↪list(stop_words_library),list(stop_words_library)], ␣
 ↪##[list(stop_words_nltk), list(stop_words_sklearn), list(stop_words_library)]
    "selecter__k":[5000],
    "vect__ngram_range":[(1,1)],
    # "vect__binary": [False]
    "vect__preprocessor": [preprocess_text,remove_punctuation,None]
    #"vect__binary": [False]

}



vectorizer = CountVectorizer()
```

```
selecter = SelectKBest(chi2)
model = LogisticRegression()
#normalizer = Normalizer()


pipe = Pipeline(
     [("vect", vectorizer),("selecter", selecter),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
The best accuracy is 93.038.
The winning parameters are {'classify__C': 10.0, 'classify__class_weight': None,
'classify__max_iter': 1000, 'classify__penalty': 'l2', 'classify__solver':
'sag', 'selecter__k': 5000, 'vect__ngram_range': (1, 1), 'vect__preprocessor':
None, 'vect__stop_words': ['most', 'through', 'everything', 'had', 'have',
'these', 'did', 'un', 'still', 'anyone', 'her', 'almost', 'mine', "hadn't",
'its', 'one', "shouldn't", 'thence', 'never', 'your', 'doing', 'out', 'three',
'some', 'due', 'below', 'although', 'wasn', 'made', 'very', 'other', 'what',
'bill', 'am', 'as', 'see', 'cant', 'whose', 'fifty', 'wherein', 'amount',
'twenty', 'nobody', 'somewhere', "you're", 'hereafter', 'along', 've', 'hence',
'against', 'hadn', 'often', 'noone', 'more', 'fifteen', 'becomes', 'seem',
'mustn', 'ltd', 'upon', 'two', "haven't", 'won', 'among', 'something', "aren't",
'them', 'do', 'then', 'yourselves', 'give', 'onto', "needn't", 'whither',
'under', 'last', "mightn't", 'seems', 'shan', "won't", 'becoming', 'therefore',
'after', 'done', 'i', 'couldnt', 'another', 'put', 'towards', 'myself', "you'd",
'yet', "shan't", 'all', 'be', 'back', 'hers', 'you', 'from', 'on', "wouldn't",
'wherever', 'not', 'y', 'if', 'because', 'become', 'such', 'so', 'an', 'co',
'once', 'move', 'several', 'ourselves', 'even', 'nowhere', 'ours', 'himself',
'toward', 're', 'hasn', 'whence', 'him', 'must', 'meanwhile', 'there', 'four',
'behind', "doesn't", 'ain', 'whereupon', 'needn', 'anything', 'where',
'together', 'well', 'everyone', 'else', 'none', 'don', 'couldn', 'take',
'should', 'than', 'anyhow', 'might', 'further', 'whatever', 'someone', 'mightn',
'who', 'thereupon', 'across', 'full', 'least', 'throughout', 'twelve', 'haven',
'being', 'namely', 'call', 'isn', 'ever', 'until', 'yours', 'will', 'inc',
```

```
"hasn't", 'm', "weren't", 'whoever', 'my', 'down', 'at', 'sometime', 'she', 't',
'herein', 'itself', 'part', 'sixty', 'here', "couldn't", 'he', 'theirs',
'whereas', 'otherwise', 'yourself', 'that', 'again', 'forty', 's', 'always',
'which', 'bottom', 'how', 'can', 'go', 'hereupon', 'since', 'just', 'latterly',
'could', 'hereby', 'll', 'mostly', "you've", 'much', 'seemed', "mustn't", 'was',
'our', 'without', 'beforehand', 'serious', 'via', 'me', 'formerly', 'why',
'enough', "should've", 'whereafter', 'perhaps', 'sincere', 'five', 'many',
'now', 'thereafter', 'about', 'detail', 'and', 'wouldn', 'cannot', 'having',
"didn't", 'it', 'eleven', 'nor', 'cry', 'either', 'thin', 'sometimes',
'seeming', 'we', 'd', 'con', 'same', 'to', 'per', 'his', 'the', 'fire', 'found',
'describe', 'already', 'within', 'whether', 'doesn', 'latter', 'has', 'therein',
'rather', 'of', 'anywhere', 'amongst', 'ten', 'o', 'would', 'front', 'de',
'alone', 'system', 'elsewhere', 'those', 'for', 'thick', 'etc', 'a', 'are',
'find', 'though', 'neither', 'whereby', 'own', 'over', 'only', 'thereby',
"don't", 'no', 'whenever', 'themselves', 'also', 'beside', 'nothing', 'thus',
'ie', 'third', 'aren', 'too', 'during', 'off', 'became', 'didn', 'fill',
'indeed', 'please', 'in', 'hasnt', 'hundred', 'afterwards', 'mill', 'name',
'their', 'former', 'but', 'moreover', 'thru', 'however', 'whole', 'been',
'next', 'besides', 'eg', 'side', "you'll", 'first', 'keep', 'somehow', 'weren',
'each', 'nevertheless', 'up', 'is', 'they', 'amoungst', 'any', 'everywhere',
'around', 'empty', "isn't", 'anyway', 'shouldn', "that'll", 'nine', 'beyond',
'while', 'whom', 'were', 'top', "she's", 'interest', 'show', 'get', 'ma',
'less', 'between', 'by', 'does', 'herself', 'few', 'above', 'into', 'with',
'six', 'may', 'except', 'eight', "wasn't", 'others', "it's", 'us', 'both',
'every', 'this', 'when', 'or', 'before']}
Run time: 114.15120077133179 seconds
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

[ ]:
```python
#initial training with stop words. 93.038

t_start = time.time()

pipe_params = {
    'classify__penalty': ['l2'],   #'classify__penalty': ['l1', 'l2'],
    'classify__C': [10.0],     #'classify__C': [0.01, 0.1, 1.0, 10.0],
    'classify__solver': ['sag'],   #'classify__solver': ['liblinear',␣
 ↪'newton-cg', 'lbfgs', 'sag', 'saga'],
    'classify__max_iter': [1000], # 'classify__max_iter': [100, 500, 1000],
    'classify__class_weight': [None, 'balanced'],     #'classify__class_weight':
 ↪ [None, 'balanced'],
    "vect__stop_words": [list(stop_words_nltk), list(stop_words_sklearn),␣
 ↪list(stop_words_library),list(stop_words_library)], ␣
 ↪##[list(stop_words_nltk), list(stop_words_sklearn), list(stop_words_library)]
    "selecter__k":[5000],
```

```
    "vect__ngram_range":[(1,1)],
    # "vect__binary": [False]
    "vect__preprocessor": [preprocess_text,remove_punctuation,None]

}



vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = LogisticRegression()
#normalizer = Normalizer()


pipe = Pipeline(
     [("vect", vectorizer),("selecter", selecter),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")



y_pred = grid.predict(test_x)
create_test_csv(y_pred,"LogisticReg.csv")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
The best accuracy is 93.037.
The winning parameters are {'classify__C': 10.0, 'classify__class_weight':
'balanced', 'classify__max_iter': 1000, 'classify__penalty': 'l2',
'classify__solver': 'sag', 'selecter__k': 5000, 'vect__ngram_range': (1, 1),
'vect__preprocessor': None, 'vect__stop_words': ['see', 'fifty', 'several',
'much', 'yet', 'often', "isn't", "shan't", 'further', 'of', 'together', 'and',
'd', "needn't", 'cannot', "aren't", 'eight', 'across', 'anything', "hadn't",
'con', 'theirs', 'once', 'anyhow', 'twelve', 'those', 'full', 'itself', 'only',
'o', 've', 'in', 'why', 'haven', 'same', 'them', 'give', 'sometime', 'behind',
'enough', 'couldnt', 'becoming', 'already', 'everywhere', 'third', 'hereupon',
```

```
'interest', 'just', 'through', 'without', 'except', 'un', 'another', 'but',
'least', 'somewhere', 're', 'perhaps', 'made', 'co', 'hasn', 'mightn', 'didn',
'onto', 'should', 'cant', 'into', 'whatever', 'from', 'since', 'six',
'wherever', 'having', 'everything', 'ltd', 'as', 'because', 'under', 'or',
'hence', 'meanwhile', 'yourself', 'bottom', 'can', 'nine', 'anyway', "mightn't",
'him', 'wasn', 'everyone', 'rather', "it's", 'becomes', 'cry', 'do', 'ever',
'hundred', 'become', 'on', 'anyone', 'then', 'most', "you've", 'will', 'keep',
'else', "haven't", 'whoever', 'being', 'during', "that'll", "she's", 'yours',
'they', 'five', 'whenever', 'seemed', 'did', 'therefore', 'get', 'call', 'up',
'ten', 'your', 'last', 'to', 'seeming', 'every', 'along', 'is', 'be', 'the',
'all', 'either', 'myself', 'never', "you'd", 'doesn', 'who', "won't",
'amoungst', 'thereby', "don't", 'whereafter', 'beyond', 'are', 'thence', 'show',
'although', 'latter', 'thereupon', 'twenty', 'something', 'his', 'side', 'had',
'somehow', 'their', 'nowhere', 'whereupon', 'ie', 'fifteen', 's', "shouldn't",
'over', 'after', 'out', 'sincere', 'someone', 'fire', 'each', 't', 'beside',
'etc', 'some', 'nobody', 'shan', "should've", 'other', 'about', 'two', 'have',
'done', 'we', 'put', 'one', 'move', 'nothing', 'more', 'yourselves', 'others',
'll', 'among', 'whereby', 'three', 'toward', 'whose', 'an', 'herself',
'towards', "you'll", 'might', 'whom', 'isn', 'these', 'though', 'whether', 'no',
'back', 'ain', 'even', 'herein', 'both', 'hereafter', 'am', 'whence', 'whereas',
'bill', 'name', 'part', 'such', 'it', 'wouldn', 'down', 'thereafter', 'if',
'she', 'don', "didn't", 'now', 'won', 'besides', 'me', 'own', 'her', 'a',
"wouldn't", 'hasnt', 'nevertheless', 'nor', 'ours', 'fill', 'he', 'does',
'there', 'between', 'take', 'again', 'not', 'please', 'four', 'almost', 'thick',
'while', 'us', 'alone', 'serious', "couldn't", 'throughout', 'top', 'could',
'therein', 'noone', 'forty', 'than', 'first', 'de', 'mine', 'latterly', 'any',
'himself', 'also', 'go', 'amount', 'wherein', 'namely', 'were', 'neither',
'find', 'has', 'before', 'at', 'less', 'may', 'elsewhere', 'couldn', 'above',
'per', 'seems', 'many', 'whole', 'still', 'been', 'so', 'around', "mustn't",
'themselves', 'here', 'hereby', 'few', 'off', 'formerly', 'thru', 'sometimes',
'was', 'i', 'eg', 'via', 'well', 'ma', 'empty', 'describe', 'mostly', 'by',
'within', 'with', 'whither', "wasn't", 'my', 'doing', 'eleven', 'for', 'upon',
'became', 'moreover', 'thin', 'would', 'below', 'always', 'former', 'mill',
'afterwards', 'too', 'seem', 'amongst', 'anywhere', 'front', 'hadn', 'needn',
'due', 'detail', 'what', 'which', 'y', 'against', 'next', 'otherwise', "you're",
'hers', 'very', 'aren', "hasn't", 'that', 'm', 'however', 'weren', 'sixty',
"weren't", 'when', 'beforehand', 'ourselves', 'where', 'you', 'indeed',
'system', "doesn't", 'inc', 'shouldn', 'thus', 'until', 'how', 'its', 'mustn',
'found', 'this', 'none', 'our', 'must']}
Run time: 110.83977627754211 seconds
File saved.

/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

[ ]:
```
#initial training with stop words. 93.038

t_start = time.time()

pipe_params = {
    'classify__penalty': ['l2'],   #'classify__penalty': ['l1', 'l2'],
    'classify__C': [10.0],     #'classify__C': [0.01, 0.1, 1.0, 10.0],
    'classify__solver': ['sag'],   #'classify__solver': ['liblinear',␣
 ↪'newton-cg', 'lbfgs', 'sag', 'saga'],
    'classify__max_iter': [10000], # 'classify__max_iter': [100, 500, 1000],
    'classify__class_weight': [None, 'balanced'],     #'classify__class_weight':
 ↪ [None, 'balanced'],
    "vect__stop_words": [list(stop_words_nltk), list(stop_words_sklearn),␣
 ↪list(stop_words_library),list(stop_words_library)], ␣
 ↪##[list(stop_words_nltk), list(stop_words_sklearn), list(stop_words_library)]
    "selecter__k":[5000],
    "vect__ngram_range":[(1,1)],
    "vect__binary": [False],


}



vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
model = LogisticRegression()
#normalizer = Normalizer()


pipe = Pipeline(
     [("vect", vectorizer),("selecter", selecter),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
#y_pred = grid.predict(test_x)
#create_test_csv(y_pred,"LogisticReg.csv")
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
The best accuracy is 92.481.
The winning parameters are {'classify__C': 10.0, 'classify__class_weight': None,
'classify__max_iter': 10000, 'classify__penalty': 'l2', 'classify__solver':
'sag', 'selecter__k': 5000, 'vect__binary': False, 'vect__ngram_range': (1, 1),
'vect__stop_words': ['see', 'fifty', 'several', 'much', 'yet', 'often', "isn't",
"shan't", 'further', 'of', 'together', 'and', 'd', "needn't", 'cannot',
"aren't", 'eight', 'across', 'anything', "hadn't", 'con', 'theirs', 'once',
'anyhow', 'twelve', 'those', 'full', 'itself', 'only', 'o', 've', 'in', 'why',
'haven', 'same', 'them', 'give', 'sometime', 'behind', 'enough', 'couldnt',
'becoming', 'already', 'everywhere', 'third', 'hereupon', 'interest', 'just',
'through', 'without', 'except', 'un', 'another', 'but', 'least', 'somewhere',
're', 'perhaps', 'made', 'co', 'hasn', 'mightn', 'didn', 'onto', 'should',
'cant', 'into', 'whatever', 'from', 'since', 'six', 'wherever', 'having',
'everything', 'ltd', 'as', 'because', 'under', 'or', 'hence', 'meanwhile',
'yourself', 'bottom', 'can', 'nine', 'anyway', "mightn't", 'him', 'wasn',
'everyone', 'rather', "it's", 'becomes', 'cry', 'do', 'ever', 'hundred',
'become', 'on', 'anyone', 'then', 'most', "you've", 'will', 'keep', 'else',
"haven't", 'whoever', 'being', 'during', "that'll", "she's", 'yours', 'they',
'five', 'whenever', 'seemed', 'did', 'therefore', 'get', 'call', 'up', 'ten',
'your', 'last', 'to', 'seeming', 'every', 'along', 'is', 'be', 'the', 'all',
'either', 'myself', 'never', "you'd", 'doesn', 'who', "won't", 'amoungst',
'thereby', "don't", 'whereafter', 'beyond', 'are', 'thence', 'show', 'although',
'latter', 'thereupon', 'twenty', 'something', 'his', 'side', 'had', 'somehow',
'their', 'nowhere', 'whereupon', 'ie', 'fifteen', 's', "shouldn't", 'over',
'after', 'out', 'sincere', 'someone', 'fire', 'each', 't', 'beside', 'etc',
'some', 'nobody', 'shan', "should've", 'other', 'about', 'two', 'have', 'done',
'we', 'put', 'one', 'move', 'nothing', 'more', 'yourselves', 'others', 'll',
'among', 'whereby', 'three', 'toward', 'whose', 'an', 'herself', 'towards',
"you'll", 'might', 'whom', 'isn', 'these', 'though', 'whether', 'no', 'back',
'ain', 'even', 'herein', 'both', 'hereafter', 'am', 'whence', 'whereas', 'bill',
'name', 'part', 'such', 'it', 'wouldn', 'down', 'thereafter', 'if', 'she',
'don', "didn't", 'now', 'won', 'besides', 'me', 'own', 'her', 'a', "wouldn't",
'hasnt', 'nevertheless', 'nor', 'ours', 'fill', 'he', 'does', 'there',
'between', 'take', 'again', 'not', 'please', 'four', 'almost', 'thick', 'while',
'us', 'alone', 'serious', "couldn't", 'throughout', 'top', 'could', 'therein',
'noone', 'forty', 'than', 'first', 'de', 'mine', 'latterly', 'any', 'himself',
'also', 'go', 'amount', 'wherein', 'namely', 'were', 'neither', 'find', 'has',
'before', 'at', 'less', 'may', 'elsewhere', 'couldn', 'above', 'per', 'seems',
'many', 'whole', 'still', 'been', 'so', 'around', "mustn't", 'themselves',
```

```
'here', 'hereby', 'few', 'off', 'formerly', 'thru', 'sometimes', 'was', 'i',
'eg', 'via', 'well', 'ma', 'empty', 'describe', 'mostly', 'by', 'within',
'with', 'whither', "wasn't", 'my', 'doing', 'eleven', 'for', 'upon', 'became',
'moreover', 'thin', 'would', 'below', 'always', 'former', 'mill', 'afterwards',
'too', 'seem', 'amongst', 'anywhere', 'front', 'hadn', 'needn', 'due', 'detail',
'what', 'which', 'y', 'against', 'next', 'otherwise', "you're", 'hers', 'very',
'aren', "hasn't", 'that', 'm', 'however', 'weren', 'sixty', "weren't", 'when',
'beforehand', 'ourselves', 'where', 'you', 'indeed', 'system', "doesn't", 'inc',
'shouldn', 'thus', 'until', 'how', 'its', 'mustn', 'found', 'this', 'none',
'our', 'must']}
Run time: 67.39258456230164 seconds
```

stacking

March 12, 2023

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from google.colab import drive
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import random
from sklearn.svm import SVC
import time
import re
import string
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2,␣
 ↪f_classif,mutual_info_classif,f_regression
from sklearn.preprocessing import Normalizer
from sklearn import model_selection
from sklearn import svm
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
```

```
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[ ]:
```
True
```

[ ]:
```
#import the data
drive.mount('/content/gdrive/', force_remount=True)

train_data_initial = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/train.
 ↪csv')
test_data = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/test.csv')

print('shape train:',train_data_initial.shape)
print('shape test:',test_data.shape)
```

```
Mounted at /content/gdrive/
shape train: (718, 2)
shape test: (279, 2)
```

[ ]:
```
def shuffle_data(df):
    random.seed(0)  # Use a fixed seed for the random number generator
    df = df.sample(frac=1, random_state=0).reset_index(drop=True)
    return df
```

[ ]:
```
#function for creating the test csv file to upload to kaggle
def create_test_csv(data, outfile_name):
  rawdata= {'subreddit':data}
```

```
  csv = pd.DataFrame(rawdata, columns = ['subreddit'])
  csv.to_csv(outfile_name,index=True, header=True)
  print ("File saved.")
```

[ ]:
```
#shuffle the data and split the features from the label
train_data = shuffle_data(train_data_initial)

train_x = train_data["body"]
train_y = train_data["subreddit"]
test_x = test_data["body"]
```

[ ]:
```
#remove punctuation
def remove_punctuation(text):
   translator = str.maketrans('', '', string.punctuation)
   text = text.translate(translator)
   return text
```

[ ]:
```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    return text
```

[ ]:
```
def print_best_params(grid):
  bestParameters = grid.best_estimator_.get_params()
  # print(bestParameters)
  for paramName in sorted(bestParameters.keys()):
    print("\t%s: %r" % (paramName, bestParameters[paramName]))
```

[ ]:
```
#create a dictionary of stop words
stop_words_nltk = set(stopwords.words('english'))
stop_words_sklearn = text.ENGLISH_STOP_WORDS
stop_words_library = stop_words_sklearn.union(stop_words_nltk)
```

[ ]:
```
#stemmer lemmatizer
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

class LemmaTokenizer_Pos:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
```

```
        return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in␣
 ↪word_tokenize(doc) if t.isalpha()]

class LemmaTokenizer:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) if t.
 ↪isalpha()]

class LemmaTokenizer_word:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) ]


class StemTokenizer:
     def __init__(self):
       self.wnl =PorterStemmer()
     def __call__(self, doc):
       return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
```

[ ]:
```
############################################################
```

[ ]:
```
stop_words_custom = [
# All pronouns and associated words
"i","i'll","i'd","i'm","i've","ive","me","myself","you","you'll","you'd","you're","you've","yo
"he'd",
"he's",
"him",
"she",
"she'll",
"she'd",
"she's",
"her",
"it",
"it'll",
"it'd",
"it's",
"itself",
"oneself",
"we",
"we'll",
"we'd",
"we're",
"we've",
```

```
"us",
"ourselves",
"they",
"they'll",
"they'd",
"they're",
"they've",
"them",
"themselves",
"everyone",
"everyone's",
"everybody",
"everybody's",
"someone",
"someone's",
"somebody",
"somebody's",
"nobody",
"nobody's",
"anyone",
"anyone's",
"everything",
"everything's",
"something",
"something's",
"nothing",
"nothing's",
"anything",
"anything's",
# All determiners and associated words
"a",
"an",
"the",
"this",
"that",
"that's",
"these",
"those",
"my",
#"mine",   #Omitted since mine can refer to something else
"your",
"yours",
"his",
"hers",
"its",
"our",
"ours",
```

```
"own",
"their",
"theirs",
"few",
"much",
"many",
"lot",
"lots",
"some",
"any",
"enough",
"all",
"both",
"half",
"either",
"neither",
"each",
"every",
"certain",
"other",
"another",
"such",
"several",
"multiple",
# "what",#Dealt with later on
"rather",
"quite",
# All prepositions
"aboard",
"about",
"above",
"across",
"after",
"against",
"along",
"amid",
"amidst",
"among",
"amongst",
"anti",
"around",
"as",
"at",
"away",
"before",
"behind",
"below",
```

```
"beneath",
"beside",
"besides",
"between",
"beyond",
"but",
"by",
"concerning",
"considering",
"despite",
"down",
"during",
"except",
"excepting",
"excluding",
"far",
"following",
"for",
"from",
"here",
"here's",
"in",
"inside",
"into",
"left",
"like",
"minus",
"near",
"of",
"off",
"on",
"onto",
"opposite",
"out",
"outside",
"over",
"past",
"per",
"plus",
"regarding",
"right",
#"round",   #Omitted
#"save",#Omitted
"since",
"than",
"there",
"there's",
```

```
"through",
"to",
"toward",
"towards",
"under",
"underneath",
"unlike",
"until",
"up",
"upon",
"versus",
"via",
"with",
"within",
"without",
# Irrelevant verbs
"may",
"might",
"will",
"won't",
"would",
"wouldn't",
"can",
"can't",
"cannot",
"could",
"couldn't",
"should",
"shouldn't",
"must",
"must've",
"be",
"being",
"been",
"am",
"are",
"aren't",
"ain't",
"is",
"isn't",
"was",
"wasn't",
"were",
"weren't",
"do",
"doing",
"don't",
```

```
"does",
"doesn't",
"did",
"didn't",
"done",
"have",
"haven't",
"having",
"has",
"hasn't",
"had",
"hadn't",
"get",
"getting",
"gets",
"got",
"gotten",
"go",
"going",
"gonna",
"goes",
"went",
"gone",
"make",
"making",
"makes",
"made",
"take",
"taking",
"takes",
"took",
"taken",
"need",
"needing",
"needs",
"needed",
"use",
"using",
"uses",
"used",
"want",
"wanna",
"wanting",
"wants",
"let",
"lets",
"letting",
```

```
"let's",
"suppose",
"supposing",
"supposes",
"supposed",
"seem",
"seeming",
"seems",
"seemed",
"say",
"saying",
"says",
"said",
"know",
"knowing",
"knows",
"knew",
"known",
"look",
"looking",
"looked",
"think",
"thinking",
"thinks",
"thought",
"feel",
"feels",
"felt",
"based",
"put",
"puts",
#"wanted"   #Omitted since the advective is relevant
# Question words and associated words
"who",
"who's",
"who've",
"who'd",
"whoever",
"whoever's",
"whom",
"whomever",
"whomever's",
"whose",
"whosever",
"whosever's",
"when",
"whenever",
```

```
"which",
"whichever",
"where",
"where's",
"where'd",
"wherever",
"why",
"why's",
"why'd",
"whyever",
"what",
"what's",
"whatever",
"whence",
"how",
"how's",
"how'd",
"however",
"whether",
"whatsoever",
# Connector words and irrelevant adverbs
"and",
"or",
"not",
"because",
"also",
"always",
"never",
"only",
"really",
"very",
"greatly",
"extremely",
"somewhat",
"no",
"nope",
"nah",
"yes",
"yep",
"yeh",
"yeah",
"maybe",
"perhaps",
"more",
"most",
"less",
"least",
```

```
"good",
"great",
"well",
"better",
"best",
"bad",
"worse",
"worst",
"too",
"thru",
"though",
"although",
"yet",
"already",
"then",
"even",
"now",
"sometimes",
"still",
"together",
"altogether",
"entirely",
"fully",
"entire",
"whole",
"completely",
"utterly",
"seemingly",
"apparently",
"clearly",
"obviously",
"actually",
"actual",
"usually",
"usual",
"literally",
"honestly",
"absolutely",
"definitely",
"generally",
"totally",
"finally",
"basically",
"essentially",
"fundamentally",
"automatically",
"immediately",
```

```
"necessarily",
"primarily",
"normally",
"perfectly",
"constantly",
"particularly",
"eventually",
"hopefully",
"mainly",
"typically",
"specifically",
"differently",
"appropriately",
"plenty",
"certainly",
"unfortunately",
"ultimately",
"unlikely",
"likely",
"potentially",
"fortunately",
"personally",
"directly",
"indirectly",
"nearly",
"closely",
"slightly",
"probably",
"possibly",
"especially",
"frequently",
"often",
"oftentimes",
"seldom",
"rarely",
"sure",
"while",
"whilst",
"able",
"unable",
"else",
"ever",
"once",
"twice",
"thrice",
"almost",
"again",
```

```
"instead",
"next",
"previous",
"unless",
"somehow",
"anyhow",
"anywhere",
"somewhere",
"everywhere",
"nowhere",
"further",
"anymore",
"later",
"ago",
"ahead",
"just",
"same",
"different",
"big",
"small",
"little",
"tiny",
"large",
"huge",
"pretty",
"mostly",
"anyway",
"anyways",
"otherwise",
"regardless",
"throughout",
"additionally",
"moreover",
"furthermore",
"meanwhile",
"afterwards",
# Irrelevant nouns
"thing",
"thing's",
"things",
"stuff",
"other's",
"others",
"another's",
"total",
"",
"false",
```

```
"none",
"way",
"kind",
# Lettered numbers and order
"zero",
"zeros",
"zeroes",
"one",
"ones",
"two",
"three",
"four",
"five",
"six",
"seven",
"eight",
"nine",
"ten",
"twenty",
"thirty",
"forty",
"fifty",
"sixty",
"seventy",
"eighty",
"ninety",
"hundred",
"hundreds",
"thousand",
"thousands",
"million",
"millions",
"first",
"last",
"second",
"third",
"fourth",
"fifth",
"sixth",
"seventh",
"eigth",
"ninth",
"tenth",
"firstly",
"secondly",
"thirdly",
"lastly",
```

```
# Greetings and slang
"hello",
"hi",
"hey",
"sup",
"yo",
"greetings",
"please",
"okay",
"ok",
"y'all",
"lol",
"rofl",
"thank",
"thanks",
"alright",
"kinda",
"dont",
"sorry",
"idk",
"tldr",
"tl",
"dr",  #This means that dr (doctor) is a bad feature because of tl;dr
"tbh",
"dude",
"tho",
"aka",
"plz",
"pls",
"bit",
"don",
# Miscellaneous
"www",
"https",
"http",
"com",
"etc"
"html",
"reddit",
"subreddit",
"subreddits",
"comments",
"reply",
"replies",
"thread",
"threads",
"post",
```

```
"posts",
"website",
"websites",
"web site",
"web sites"]
print('length custom:',len(stop_words_custom))
```

```
length custom: 589
```

[ ]:
```
#base condition with stacking
from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer

# Define the base estimators for the stacking classifier
estimators = [
    ('lr', LogisticRegression(random_state=42)),
    ('mnb', MultinomialNB())
]

# Define the stacking classifier pipeline
stacking_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('stacking', StackingClassifier(estimators=estimators))
])

# Define the grid search parameters
params = {
   # 'tfidf__max_df': [0.5, 0.75, 1.0],
  "tfidf__stop_words": [list(stop_words_library)],
   # 'tfidf__ngram_range': [(1,1), (1,2), (1,3)],
   # 'stacking__final_estimator__penalty': ['l1', 'l2'],
   # 'stacking__final_estimator__C': [0.1, 1.0, 10.0],
   # 'stacking__final_estimator__solver': ['liblinear', 'lbfgs']
}

# Define the grid search object
grid_search = GridSearchCV(stacking_pipeline, params, cv=5,scoring='accuracy')

# Fit the grid search object to the training data
grid_search.fit(train_x, train_y)

#accuracy = round(grid.best_score_ * 100,3)
accuracy = round(grid_search.best_score_ * 100,3)
```

```
print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid_search.best_params_}")
#print(f"Run time: {elapsed_time} seconds")
```

```
The best accuracy is 94.568.
The winning parameters are {'tfidf__stop_words': ['even', 'put', 'eleven',
'won', 'didn', 'beforehand', 'toward', 'couldnt', 'mostly', 'eight', 'either',
'enough', 'your', 'while', 'been', 'anyway', 'sincere', 'hasnt', 'others',
'another', 'none', 'itself', 'as', 'formerly', 'often', 'about', 'off', 'just',
'during', 't', 'cannot', 'rather', "aren't", 'too', 'ever', 'wasn', 'less',
'yourself', 'myself', 'do', 'hereafter', "that'll", 'became', 'will', 'back',
"haven't", 'seemed', 'name', 'one', 'never', 'so', 'onto', "wasn't", 'find',
'until', 'if', "won't", 'here', 'elsewhere', 'no', 'those', 'needn', 'hence',
'meanwhile', 'from', 'hereupon', 'for', 'almost', 'did', 'least', 'with', 'she',
'many', 'without', 'noone', 'thereupon', 'not', 'my', 'throughout', 'thick',
'such', 'hadn', 'us', 'all', 'now', 'twenty', 'once', 'at', 'fifty', 'anywhere',
'whereas', 'former', 'else', 'always', 'sometimes', 'please', "mightn't",
'mightn', 'd', 'same', 'other', 'few', 'nobody', 'describe', 'sometime',
'somewhere', 'etc', 'seem', 'seems', "needn't", 'mill', 'which', 'thereafter',
'sixty', 'together', 'therein', 'two', "shan't", 'between', 'he', 'thin',
'already', 'his', 'their', 'hereby', 'doing', 'indeed', 'first', 'latterly',
'still', 'or', 'm', 'nor', 'can', 'neither', "hasn't", 'll', 'next', 'when',
'thru', 'over', 'hers', 'mustn', 'besides', 'could', 'side', 'ten',
'yourselves', 'move', 'nevertheless', 'ours', 'this', 'perhaps', 'fifteen',
"it's", 'well', 'con', 'up', 'un', 'be', 'mine', 'around', 'has', 'whatever',
'wouldn', 'them', 'five', 'last', 'each', "you're", 'nowhere', 'shouldn',
'wherever', 'ie', 'anyone', 'again', 'were', 'via', 'theirs', 'being', 'anyhow',
'it', 'more', 'under', 'have', 'since', 'through', 'having', "you'll", 'four',
'whereby', 'anything', 'front', 'afterwards', 'a', 'does', 's', 'six',
'somehow', 'should', 'shan', 'would', 'its', 'isn', 'any', 'where', 'keep',
'per', 'also', 'among', 'only', 'except', 'must', 'though', 'take', 'amoungst',
'behind', "isn't", 'of', 'done', 'show', 'own', 'by', "shouldn't", "weren't",
'give', 'after', 'twelve', "don't", 'thence', "wouldn't", "you've", 'then',
'these', 'to', 'everything', 'namely', "you'd", 'beside', 'i', 'ltd', 'don',
'me', 'due', "hadn't", 'hasn', 'made', 'whoever', 'above', 'forty',
'themselves', 'both', 'hundred', 're', 'our', 'amongst', 'however', 'moreover',
'out', 'fill', "couldn't", 'down', 'whom', 'become', 'haven', 'weren', 'thus',
'ma', 'below', 'becomes', 'everywhere', 'interest', 'much', 'herein', 'yours',
'seeming', 'is', 'nine', 'full', 'ourselves', 'ain', 'latter', 'across', 'am',
'call', 'whereupon', 'something', "doesn't", 'found', 'why', 'most',
'therefore', 'co', 'thereby', 'someone', 'empty', 'on', 'who', 'towards',
'whereafter', 'go', 'there', 'cry', 'they', 'because', 'beyond', 'bottom',
'that', 'de', 'further', 'y', 'very', 'whole', 'get', 'alone', 'than', 'detail',
'and', 'part', 'whenever', 'top', 'every', 'him', 'but', 'amount', 'everyone',
'herself', 'aren', 'along', 'three', 'fire', 'against', 'we', "she's",
'becoming', 've', 'are', 'bill', 'before', "mustn't", 'within', 'wherein',
```

```
'doesn', 'was', 'nothing', 'himself', 'the', 'whence', 'whither', 'otherwise',
'serious', 'eg', 'in', 'inc', 'into', 'o', 'some', 'upon', 'whether', 'yet',
'cant', 'several', 'how', 'had', 'may', 'whose', "should've", 'system',
"didn't", 'an', 'third', 'her', 'see', 'couldn', 'although', 'you', 'might',
'what']}
```

[ ]:
```python
#base condition with stacking
#=>94.846
from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer

# Define the base estimators for the stacking classifier
estimators = [
    ('lr', LogisticRegression(random_state=42)),
    ('mnb', MultinomialNB())
]

# Define the stacking classifier pipeline
stacking_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('stacking', StackingClassifier(estimators=estimators))
])

# Define the grid search parameters
params = {
   # 'tfidf__max_df': [0.5, 0.75, 1.0],
  "tfidf__stop_words": [list(stop_words_library),list(stop_words_custom)],
   # 'tfidf__ngram_range': [(1,1), (1,2), (1,3)],
   # 'stacking__final_estimator__penalty': ['l1', 'l2'],
   # 'stacking__final_estimator__C': [0.1, 1.0, 10.0],
   # 'stacking__final_estimator__solver': ['liblinear', 'lbfgs']
}

# Define the grid search object
grid_search = GridSearchCV(stacking_pipeline, params, cv=5,scoring='accuracy'␣
 ↪,verbose=1, n_jobs=-1)

# Fit the grid search object to the training data
grid_search.fit(train_x, train_y)

#accuracy = round(grid.best_score_ * 100,3)
accuracy = round(grid_search.best_score_ * 100,3)
```

```
print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid_search.best_params_}")
#print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(
```

```
The best accuracy is 94.846.
The winning parameters are {'tfidf__stop_words': ['i', "i'll", "i'd", "i'm",
"i've", 'ive', 'me', 'myself', 'you', "you'll", "you'd", "you're", "you've",
'yourself', 'he', "he'll", "he'd", "he's", 'him', 'she', "she'll", "she'd",
"she's", 'her', 'it', "it'll", "it'd", "it's", 'itself', 'oneself', 'we',
"we'll", "we'd", "we're", "we've", 'us', 'ourselves', 'they', "they'll",
"they'd", "they're", "they've", 'them', 'themselves', 'everyone', "everyone's",
'everybody', "everybody's", 'someone', "someone's", 'somebody', "somebody's",
'nobody', "nobody's", 'anyone', "anyone's", 'everything', "everything's",
'something', "something's", 'nothing', "nothing's", 'anything', "anything's",
'a', 'an', 'the', 'this', 'that', "that's", 'these', 'those', 'my', 'your',
'yours', 'his', 'hers', 'its', 'our', 'ours', 'own', 'their', 'theirs', 'few',
'much', 'many', 'lot', 'lots', 'some', 'any', 'enough', 'all', 'both', 'half',
'either', 'neither', 'each', 'every', 'certain', 'other', 'another', 'such',
'several', 'multiple', 'rather', 'quite', 'aboard', 'about', 'above', 'across',
'after', 'against', 'along', 'amid', 'amidst', 'among', 'amongst', 'anti',
'around', 'as', 'at', 'away', 'before', 'behind', 'below', 'beneath', 'beside',
'besides', 'between', 'beyond', 'but', 'by', 'concerning', 'considering',
'despite', 'down', 'during', 'except', 'excepting', 'excluding', 'far',
'following', 'for', 'from', 'here', "here's", 'in', 'inside', 'into', 'left',
'like', 'minus', 'near', 'of', 'off', 'on', 'onto', 'opposite', 'out',
'outside', 'over', 'past', 'per', 'plus', 'regarding', 'right', 'since', 'than',
'there', "there's", 'through', 'to', 'toward', 'towards', 'under', 'underneath',
'unlike', 'until', 'up', 'upon', 'versus', 'via', 'with', 'within', 'without',
'may', 'might', 'will', "won't", 'would', "wouldn't", 'can', "can't", 'cannot',
'could', "couldn't", 'should', "shouldn't", 'must', "must've", 'be', 'being',
'been', 'am', 'are', "aren't", "ain't", 'is', "isn't", 'was', "wasn't", 'were',
"weren't", 'do', 'doing', "don't", 'does', "doesn't", 'did', "didn't", 'done',
'have', "haven't", 'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting',
'gets', 'got', 'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make',
'making', 'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need',
'needing', 'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna',
'wanting', 'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing',
'supposes', 'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying',
'says', 'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
```

```
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites']}
```

[ ]:
```
#base condition with stacking

from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer

# Define the base estimators for the stacking classifier
estimators = [
    ('lr', LogisticRegression(random_state=42)),
    ('mnb', MultinomialNB())
]

# Define the stacking classifier pipeline
stacking_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    ('stacking', StackingClassifier(estimators=estimators))
])

# Define the grid search parameters
params = {
   # 'tfidf__max_df': [0.5, 0.75, 1.0],
  "cv__stop_words": [list(stop_words_custom)],
    'stacking__mnb__alpha': [0.0001, 0.001, 0.01,0.5],
   # 'tfidf__ngram_range': [(1,1), (1,2), (1,3)],
   #'stacking__final_estimator__penalty': ['l1', 'l2'],
   # 'stacking__final_estimator__C': [0.1, 1.0, 10.0],
   # 'stacking__final_estimator__solver': ['liblinear', 'lbfgs']
}

# Define the grid search object
grid_search = GridSearchCV(stacking_pipeline, params, cv=5,scoring='accuracy'␣
 ↪,verbose=1, n_jobs=-1)

# Fit the grid search object to the training data
grid_search.fit(train_x, train_y)

#accuracy = round(grid.best_score_ * 100,3)
accuracy = round(grid_search.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid_search.best_params_}")
#print(f"Run time: {elapsed_time} seconds")

#print_best_params(grid_search)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
```

```
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 95.123.
The winning parameters are {'cv__stop_words': ['i', "i'll", "i'd", "i'm",
"i've", 'ive', 'me', 'myself', 'you', "you'll", "you'd", "you're", "you've",
'yourself', 'he', "he'll", "he'd", "he's", 'him', 'she', "she'll", "she'd",
"she's", 'her', 'it', "it'll", "it'd", "it's", 'itself', 'oneself', 'we',
"we'll", "we'd", "we're", "we've", 'us', 'ourselves', 'they', "they'll",
"they'd", "they're", "they've", 'them', 'themselves', 'everyone', "everyone's",
'everybody', "everybody's", 'someone', "someone's", 'somebody', "somebody's",
'nobody', "nobody's", 'anyone', "anyone's", 'everything', "everything's",
'something', "something's", 'nothing', "nothing's", 'anything', "anything's",
'a', 'an', 'the', 'this', 'that', "that's", 'these', 'those', 'my', 'your',
'yours', 'his', 'hers', 'its', 'our', 'ours', 'own', 'their', 'theirs', 'few',
'much', 'many', 'lot', 'lots', 'some', 'any', 'enough', 'all', 'both', 'half',
'either', 'neither', 'each', 'every', 'certain', 'other', 'another', 'such',
'several', 'multiple', 'rather', 'quite', 'aboard', 'about', 'above', 'across',
'after', 'against', 'along', 'amid', 'amidst', 'among', 'amongst', 'anti',
'around', 'as', 'at', 'away', 'before', 'behind', 'below', 'beneath', 'beside',
'besides', 'between', 'beyond', 'but', 'by', 'concerning', 'considering',
'despite', 'down', 'during', 'except', 'excepting', 'excluding', 'far',
'following', 'for', 'from', 'here', "here's", 'in', 'inside', 'into', 'left',
'like', 'minus', 'near', 'of', 'off', 'on', 'onto', 'opposite', 'out',
'outside', 'over', 'past', 'per', 'plus', 'regarding', 'right', 'since', 'than',
'there', "there's", 'through', 'to', 'toward', 'towards', 'under', 'underneath',
'unlike', 'until', 'up', 'upon', 'versus', 'via', 'with', 'within', 'without',
'may', 'might', 'will', "won't", 'would', "wouldn't", 'can', "can't", 'cannot',
'could', "couldn't", 'should', "shouldn't", 'must', "must've", 'be', 'being',
'been', 'am', 'are', "aren't", "ain't", 'is', "isn't", 'was', "wasn't", 'were',
"weren't", 'do', 'doing', "don't", 'does', "doesn't", 'did', "didn't", 'done',
'have', "haven't", 'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting',
'gets', 'got', 'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make',
'making', 'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need',
'needing', 'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna',
'wanting', 'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing',
'supposes', 'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying',
'says', 'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
```

```
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites'],
'stacking__mnb__alpha': 0.5}
```

[ ]:
```python
#base condition with stacking

from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer

# Define the base estimators for the stacking classifier
estimators = [
    ('lr', LogisticRegression(random_state=42)),
    ('mnb', MultinomialNB())
```

```
]

selecter = SelectKBest(chi2)
normalizer = Normalizer()


# Define the stacking classifier pipeline
stacking_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    #("selecter", selecter),
    ("normalizer",normalizer),
    ('stacking', StackingClassifier(estimators=estimators))
])

# Define the grid search parameters
params = {
   # 'tfidf__max_df': [0.5, 0.75, 1.0],
    'stacking__mnb__alpha': [0.5],
       # "selecter__k":[5000],
          "cv__stop_words": [list(stop_words_custom)],
            "normalizer__norm": ['l2','l1']
   # 'tfidf__ngram_range': [(1,1), (1,2), (1,3)],
   #'stacking__final_estimator__penalty': ['l1', 'l2'],
   # 'stacking__final_estimator__C': [0.1, 1.0, 10.0],
   # 'stacking__final_estimator__solver': ['liblinear', 'lbfgs']
}

# Define the grid search object
grid_search = GridSearchCV(stacking_pipeline, params, cv=5,scoring='accuracy'␣
 ↪,verbose=1, n_jobs=-1)

# Fit the grid search object to the training data
grid_search.fit(train_x, train_y)

#accuracy = round(grid.best_score_ * 100,3)
accuracy = round(grid_search.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid_search.best_params_}")
#print(f"Run time: {elapsed_time} seconds")

#print_best_params(grid_search)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
```

```
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 95.123.
The winning parameters are {'cv__stop_words': ['i', "i'll", "i'd", "i'm",
"i've", 'ive', 'me', 'myself', 'you', "you'll", "you'd", "you're", "you've",
'yourself', 'he', "he'll", "he'd", "he's", 'him', 'she', "she'll", "she'd",
"she's", 'her', 'it', "it'll", "it'd", "it's", 'itself', 'oneself', 'we',
"we'll", "we'd", "we're", "we've", 'us', 'ourselves', 'they', "they'll",
"they'd", "they're", "they've", 'them', 'themselves', 'everyone', "everyone's",
'everybody', "everybody's", 'someone', "someone's", 'somebody', "somebody's",
'nobody', "nobody's", 'anyone', "anyone's", 'everything', "everything's",
'something', "something's", 'nothing', "nothing's", 'anything', "anything's",
'a', 'an', 'the', 'this', 'that', "that's", 'these', 'those', 'my', 'your',
'yours', 'his', 'hers', 'its', 'our', 'ours', 'own', 'their', 'theirs', 'few',
'much', 'many', 'lot', 'lots', 'some', 'any', 'enough', 'all', 'both', 'half',
'either', 'neither', 'each', 'every', 'certain', 'other', 'another', 'such',
'several', 'multiple', 'rather', 'quite', 'aboard', 'about', 'above', 'across',
'after', 'against', 'along', 'amid', 'amidst', 'among', 'amongst', 'anti',
'around', 'as', 'at', 'away', 'before', 'behind', 'below', 'beneath', 'beside',
'besides', 'between', 'beyond', 'but', 'by', 'concerning', 'considering',
'despite', 'down', 'during', 'except', 'excepting', 'excluding', 'far',
'following', 'for', 'from', 'here', "here's", 'in', 'inside', 'into', 'left',
'like', 'minus', 'near', 'of', 'off', 'on', 'onto', 'opposite', 'out',
'outside', 'over', 'past', 'per', 'plus', 'regarding', 'right', 'since', 'than',
'there', "there's", 'through', 'to', 'toward', 'towards', 'under', 'underneath',
'unlike', 'until', 'up', 'upon', 'versus', 'via', 'with', 'within', 'without',
'may', 'might', 'will', "won't", 'would', "wouldn't", 'can', "can't", 'cannot',
'could', "couldn't", 'should', "shouldn't", 'must', "must've", 'be', 'being',
'been', 'am', 'are', "aren't", "ain't", 'is', "isn't", 'was', "wasn't", 'were',
"weren't", 'do', 'doing', "don't", 'does', "doesn't", 'did', "didn't", 'done',
'have', "haven't", 'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting',
'gets', 'got', 'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make',
'making', 'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need',
'needing', 'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna',
'wanting', 'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing',
'supposes', 'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying',
'says', 'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
```

```
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites'], 'normalizer__norm':
'l2', 'stacking__mnb__alpha': 0.5}
```

[ ]:
```
#base condition with stacking

from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
#    'stacking__lr__solver': ['lbfgs', 'liblinear', 'newton-cg',␣
 ↪'newton-cholesky', 'sag', 'saga'],


# Define the base estimators for the stacking classifier
```

```
estimators = [
    ('lr', LogisticRegression(random_state=42)),
    ('mnb', MultinomialNB())
]

selecter = SelectKBest(chi2)
normalizer = Normalizer()


# Define the stacking classifier pipeline
stacking_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    #("selecter", selecter),
    ("normalizer",normalizer),
    ('stacking', StackingClassifier(estimators=estimators))
])

# Define the grid search parameters
params = {
   # 'tfidf__max_df': [0.5, 0.75, 1.0],
    'stacking__mnb__alpha': [0.5],
       # "selecter__k":[5000],
    "cv__stop_words": [list(stop_words_custom)],
    "normalizer__norm": ['l2','l1'],
   # 'tfidf__ngram_range': [(1,1), (1,2), (1,3)],
    'stacking__lr__solver': ['sag', 'saga'],
}

# Define the grid search object
grid_search = GridSearchCV(stacking_pipeline, params, cv=5,scoring='accuracy'␣
 ↪,verbose=1, n_jobs=-1)

# Fit the grid search object to the training data
grid_search.fit(train_x, train_y)

#accuracy = round(grid.best_score_ * 100,3)
accuracy = round(grid_search.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid_search.best_params_}")
#print(f"Run time: {elapsed_time} seconds")

#print_best_params(grid_search)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
```

```
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 95.123.
The winning parameters are {'cv__stop_words': ['i', "i'll", "i'd", "i'm",
"i've", 'ive', 'me', 'myself', 'you', "you'll", "you'd", "you're", "you've",
'yourself', 'he', "he'll", "he'd", "he's", 'him', 'she', "she'll", "she'd",
"she's", 'her', 'it', "it'll", "it'd", "it's", 'itself', 'oneself', 'we',
"we'll", "we'd", "we're", "we've", 'us', 'ourselves', 'they', "they'll",
"they'd", "they're", "they've", 'them', 'themselves', 'everyone', "everyone's",
'everybody', "everybody's", 'someone', "someone's", 'somebody', "somebody's",
'nobody', "nobody's", 'anyone', "anyone's", 'everything', "everything's",
'something', "something's", 'nothing', "nothing's", 'anything', "anything's",
'a', 'an', 'the', 'this', 'that', "that's", 'these', 'those', 'my', 'your',
'yours', 'his', 'hers', 'its', 'our', 'ours', 'own', 'their', 'theirs', 'few',
'much', 'many', 'lot', 'lots', 'some', 'any', 'enough', 'all', 'both', 'half',
'either', 'neither', 'each', 'every', 'certain', 'other', 'another', 'such',
'several', 'multiple', 'rather', 'quite', 'aboard', 'about', 'above', 'across',
'after', 'against', 'along', 'amid', 'amidst', 'among', 'amongst', 'anti',
'around', 'as', 'at', 'away', 'before', 'behind', 'below', 'beneath', 'beside',
'besides', 'between', 'beyond', 'but', 'by', 'concerning', 'considering',
'despite', 'down', 'during', 'except', 'excepting', 'excluding', 'far',
'following', 'for', 'from', 'here', "here's", 'in', 'inside', 'into', 'left',
'like', 'minus', 'near', 'of', 'off', 'on', 'onto', 'opposite', 'out',
'outside', 'over', 'past', 'per', 'plus', 'regarding', 'right', 'since', 'than',
'there', "there's", 'through', 'to', 'toward', 'towards', 'under', 'underneath',
'unlike', 'until', 'up', 'upon', 'versus', 'via', 'with', 'within', 'without',
'may', 'might', 'will', "won't", 'would', "wouldn't", 'can', "can't", 'cannot',
'could', "couldn't", 'should', "shouldn't", 'must', "must've", 'be', 'being',
'been', 'am', 'are', "aren't", "ain't", 'is', "isn't", 'was', "wasn't", 'were',
"weren't", 'do', 'doing', "don't", 'does', "doesn't", 'did', "didn't", 'done',
'have', "haven't", 'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting',
'gets', 'got', 'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make',
'making', 'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need',
'needing', 'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna',
'wanting', 'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing',
'supposes', 'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying',
'says', 'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
```

```
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites'], 'normalizer__norm':
'l2', 'stacking__lr__solver': 'sag', 'stacking__mnb__alpha': 0.5}
```

[ ]:
```
#base condition with stacking

from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
#    'stacking__lr__solver': ['lbfgs', 'liblinear', 'newton-cg',␣
 ↪'newton-cholesky', 'sag', 'saga'],
```

```
# Define the base estimators for the stacking classifier
estimators = [
    ('lr', LogisticRegression(random_state=42)),
    ('mnb', MultinomialNB())
]

selecter = SelectKBest(chi2)
normalizer = Normalizer()


# Define the stacking classifier pipeline
stacking_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    #("selecter", selecter),
    ("normalizer",normalizer),
    ('stacking', StackingClassifier(estimators=estimators))
])

# Define the grid search parameters
params = {
   # 'tfidf__max_df': [0.5, 0.75, 1.0],
    'stacking__mnb__alpha': [0.5],
       # "selecter__k":[5000],
    "cv__stop_words": [list(stop_words_custom)],
    "normalizer__norm": ['l2','l1'],
   # 'tfidf__ngram_range': [(1,1), (1,2), (1,3)],
    'stacking__lr__solver': ['sag', 'lbfgs'],
}

# Define the grid search object
grid_search = GridSearchCV(stacking_pipeline, params, cv=5,scoring='accuracy',
 ↪verbose=1, n_jobs=-1)

# Fit the grid search object to the training data
grid_search.fit(train_x, train_y)

#accuracy = round(grid.best_score_ * 100,3)
accuracy = round(grid_search.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid_search.best_params_}")
#print(f"Run time: {elapsed_time} seconds")

#print_best_params(grid_search)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
```

```
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 95.123.
The winning parameters are {'cv__stop_words': ['i', "i'll", "i'd", "i'm",
"i've", 'ive', 'me', 'myself', 'you', "you'll", "you'd", "you're", "you've",
'yourself', 'he', "he'll", "he'd", "he's", 'him', 'she', "she'll", "she'd",
"she's", 'her', 'it', "it'll", "it'd", "it's", 'itself', 'oneself', 'we',
"we'll", "we'd", "we're", "we've", 'us', 'ourselves', 'they', "they'll",
"they'd", "they're", "they've", 'them', 'themselves', 'everyone', "everyone's",
'everybody', "everybody's", 'someone', "someone's", 'somebody', "somebody's",
'nobody', "nobody's", 'anyone', "anyone's", 'everything', "everything's",
'something', "something's", 'nothing', "nothing's", 'anything', "anything's",
'a', 'an', 'the', 'this', 'that', "that's", 'these', 'those', 'my', 'your',
'yours', 'his', 'hers', 'its', 'our', 'ours', 'own', 'their', 'theirs', 'few',
'much', 'many', 'lot', 'lots', 'some', 'any', 'enough', 'all', 'both', 'half',
'either', 'neither', 'each', 'every', 'certain', 'other', 'another', 'such',
'several', 'multiple', 'rather', 'quite', 'aboard', 'about', 'above', 'across',
'after', 'against', 'along', 'amid', 'amidst', 'among', 'amongst', 'anti',
'around', 'as', 'at', 'away', 'before', 'behind', 'below', 'beneath', 'beside',
'besides', 'between', 'beyond', 'but', 'by', 'concerning', 'considering',
'despite', 'down', 'during', 'except', 'excepting', 'excluding', 'far',
'following', 'for', 'from', 'here', "here's", 'in', 'inside', 'into', 'left',
'like', 'minus', 'near', 'of', 'off', 'on', 'onto', 'opposite', 'out',
'outside', 'over', 'past', 'per', 'plus', 'regarding', 'right', 'since', 'than',
'there', "there's", 'through', 'to', 'toward', 'towards', 'under', 'underneath',
'unlike', 'until', 'up', 'upon', 'versus', 'via', 'with', 'within', 'without',
'may', 'might', 'will', "won't", 'would', "wouldn't", 'can', "can't", 'cannot',
'could', "couldn't", 'should', "shouldn't", 'must', "must've", 'be', 'being',
'been', 'am', 'are', "aren't", "ain't", 'is', "isn't", 'was', "wasn't", 'were',
"weren't", 'do', 'doing', "don't", 'does', "doesn't", 'did', "didn't", 'done',
'have', "haven't", 'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting',
'gets', 'got', 'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make',
'making', 'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need',
'needing', 'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna',
'wanting', 'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing',
'supposes', 'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying',
'says', 'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
```

```
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites'], 'normalizer__norm':
'l2', 'stacking__lr__solver': 'sag', 'stacking__mnb__alpha': 0.5}
```

[ ]:
```
#base condition with stacking

from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
#    'stacking__lr__solver': ['lbfgs', 'liblinear', 'newton-cg',␣
 ↪'newton-cholesky', 'sag', 'saga'],
```

```
# Define the base estimators for the stacking classifier
estimators = [
    ('lr', LogisticRegression(random_state=42)),
    ('mnb', MultinomialNB())
]

selecter = SelectKBest(chi2)
normalizer = Normalizer()


# Define the stacking classifier pipeline
stacking_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    ("selecter", selecter),
    ("normalizer",normalizer),
    ('stacking', StackingClassifier(estimators=estimators))
])

# Define the grid search parameters
params = {
    'cv__max_df': [0.5, 0.75],
    'stacking__mnb__alpha': [0.5],
    "selecter__k":[5000],
    "cv__stop_words": [list(stop_words_custom)],
    'cv__preprocessor': [preprocess_text],
    "normalizer__norm": ['l2'],
   'cv__ngram_range': [(1,1)],
    'stacking__lr__solver': ['sag'],
}

grid_search = GridSearchCV(stacking_pipeline, params, cv=5,scoring='accuracy'␣
 ↪,verbose=1, n_jobs=-1)

grid_search.fit(train_x, train_y)

accuracy = round(grid_search.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid_search.best_params_}")

#print_best_params(grid_search)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
```

```
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 95.123.
The winning parameters are {'cv__max_df': 0.75, 'cv__ngram_range': (1, 1),
'cv__preprocessor': <function preprocess_text at 0x7f6cc558ea60>,
'cv__stop_words': ['i', "i'll", "i'd", "i'm", "i've", 'ive', 'me', 'myself',
'you', "you'll", "you'd", "you're", "you've", 'yourself', 'he', "he'll", "he'd",
"he's", 'him', 'she', "she'll", "she'd", "she's", 'her', 'it', "it'll", "it'd",
"it's", 'itself', 'oneself', 'we', "we'll", "we'd", "we're", "we've", 'us',
'ourselves', 'they', "they'll", "they'd", "they're", "they've", 'them',
'themselves', 'everyone', "everyone's", 'everybody', "everybody's", 'someone',
"someone's", 'somebody', "somebody's", 'nobody', "nobody's", 'anyone',
"anyone's", 'everything', "everything's", 'something', "something's", 'nothing',
"nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this', 'that',
"that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its', 'our',
'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots', 'some',
'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each', 'every',
'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather', 'quite',
'aboard', 'about', 'above', 'across', 'after', 'against', 'along', 'amid',
'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away', 'before',
'behind', 'below', 'beneath', 'beside', 'besides', 'between', 'beyond', 'but',
'by', 'concerning', 'considering', 'despite', 'down', 'during', 'except',
'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here', "here's",
'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off', 'on',
'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
```

```
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites'], 'normalizer__norm':
'l2', 'selecter__k': 5000, 'stacking__lr__solver': 'sag',
'stacking__mnb__alpha': 0.5}
```

[ ]:
```python
#final

from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
# Define the base estimators for the stacking classifier
final_estimators = [
    ('lr', LogisticRegression(random_state=42)),
    ('mnb', MultinomialNB())
]

final_selecter = SelectKBest(chi2)
final_normalizer = Normalizer()


# Define the stacking classifier pipeline
final_stacking_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    ("selecter", final_selecter),
    ("normalizer",final_normalizer),
    ('stacking', StackingClassifier(estimators=final_estimators))
])

# Define the grid search parameters
final_params = {
    'cv__max_df': [0.5, 0.75],
    'stacking__mnb__alpha': [0.5],
    "selecter__k":[5000],
    "cv__stop_words": [list(stop_words_custom)],
    'cv__preprocessor': [preprocess_text],
    "normalizer__norm": ['l2'],
   'cv__ngram_range': [(1,1)],
    'stacking__lr__solver': ['sag'],
}

final_grid = GridSearchCV(final_stacking_pipeline, final_params,␣
 ↪cv=5,scoring='accuracy' ,verbose=1, n_jobs=-1)

final_grid.fit(train_x, train_y)

accuracy = round(final_grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {final_grid.best_params_}")

#print_best_params(grid_search)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
```

```
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 95.123.
The winning parameters are {'cv__max_df': 0.75, 'cv__ngram_range': (1, 1),
'cv__preprocessor': <function preprocess_text at 0x7f6cc558ea60>,
'cv__stop_words': ['i', "i'll", "i'd", "i'm", "i've", 'ive', 'me', 'myself',
'you', "you'll", "you'd", "you're", "you've", 'yourself', 'he', "he'll", "he'd",
"he's", 'him', 'she', "she'll", "she'd", "she's", 'her', 'it', "it'll", "it'd",
"it's", 'itself', 'oneself', 'we', "we'll", "we'd", "we're", "we've", 'us',
'ourselves', 'they', "they'll", "they'd", "they're", "they've", 'them',
'themselves', 'everyone', "everyone's", 'everybody', "everybody's", 'someone',
"someone's", 'somebody', "somebody's", 'nobody', "nobody's", 'anyone',
"anyone's", 'everything', "everything's", 'something', "something's", 'nothing',
"nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this', 'that',
"that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its', 'our',
'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots', 'some',
'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each', 'every',
'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather', 'quite',
'aboard', 'about', 'above', 'across', 'after', 'against', 'along', 'amid',
'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away', 'before',
'behind', 'below', 'beneath', 'beside', 'besides', 'between', 'beyond', 'but',
'by', 'concerning', 'considering', 'despite', 'down', 'during', 'except',
'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here', "here's",
'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off', 'on',
'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
```

```
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites'], 'normalizer__norm':
'l2', 'selecter__k': 5000, 'stacking__lr__solver': 'sag',
'stacking__mnb__alpha': 0.5}
```

[ ]:
```
y_pred_new = final_grid.predict(test_x)
create_test_csv(y_pred_new,"Stacking_MultiNB-Logistic-05032023_01.csv")
```

```
File saved.
```

[ ]:
```
############################################################## CNN & Multinomial
```

[ ]:
```
def print_best_params(grid):
  bestParameters = grid.best_estimator_.get_params()
  # print(bestParameters)
  for paramName in sorted(bestParameters.keys()):
```

```
    print("\t%s: %r" % (paramName, bestParameters[paramName]))
```

[ ]:
```
#new ensemble
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score



# Define the base estimators for the stacking classifier
ensemble_estimators = [('nb', MultinomialNB()), ('mlp', MLPClassifier())]

ensemble_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    ('stacking', StackingClassifier(estimators=ensemble_estimators))
])

# Define the grid search parameters
ensemble_params = {
    #'cv__max_df': [0.5, 0.75, 1.0],
   # 'nb__alpha': [0.1, 0.5, 1.0],
    "cv__stop_words": [list(stop_words_custom)],
   # 'mlp__hidden_layer_sizes': [(50,), (100,), (200,)],
    'stacking__mlp__alpha': [0.1],
}

ensemble_grid = GridSearchCV(ensemble_pipeline, ensemble_params,␣
 ↪cv=5,scoring='accuracy' ,verbose=1, n_jobs=-1)

ensemble_grid.fit(train_x, train_y)

accuracy = round(ensemble_grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {ensemble_grid.best_params_}")

print_best_params(ensemble_grid)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
/usr/local/lib/python3.8/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
```

```
  warnings.warn(
/usr/local/lib/python3.8/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/usr/local/lib/python3.8/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/usr/local/lib/python3.8/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/usr/local/lib/python3.8/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
The best accuracy is 93.729.
The winning parameters are {'stacking__mlp__alpha': 0.1}
        cv: TfidfVectorizer()
        cv__analyzer: 'word'
        cv__binary: False
        cv__decode_error: 'strict'
        cv__dtype: <class 'numpy.float64'>
        cv__encoding: 'utf-8'
        cv__input: 'content'
        cv__lowercase: True
        cv__max_df: 1.0
        cv__max_features: None
        cv__min_df: 1
        cv__ngram_range: (1, 1)
        cv__norm: 'l2'
        cv__preprocessor: None
        cv__smooth_idf: True
        cv__stop_words: None
        cv__strip_accents: None
        cv__sublinear_tf: False
        cv__token_pattern: '(?u)\\b\\w\\w+\\b'
        cv__tokenizer: None
        cv__use_idf: True
        cv__vocabulary: None
        memory: None
        stacking: StackingClassifier(estimators=[('nb', MultinomialNB()),
```

```
                              ('mlp', MLPClassifier(alpha=0.1))])
        stacking__cv: None
        stacking__estimators: [('nb', MultinomialNB()), ('mlp',
MLPClassifier(alpha=0.1))]
        stacking__final_estimator: None
        stacking__mlp: MLPClassifier(alpha=0.1)
        stacking__mlp__activation: 'relu'
        stacking__mlp__alpha: 0.1
        stacking__mlp__batch_size: 'auto'
        stacking__mlp__beta_1: 0.9
        stacking__mlp__beta_2: 0.999
        stacking__mlp__early_stopping: False
        stacking__mlp__epsilon: 1e-08
        stacking__mlp__hidden_layer_sizes: (100,)
        stacking__mlp__learning_rate: 'constant'
        stacking__mlp__learning_rate_init: 0.001
        stacking__mlp__max_fun: 15000
        stacking__mlp__max_iter: 200
        stacking__mlp__momentum: 0.9
        stacking__mlp__n_iter_no_change: 10
        stacking__mlp__nesterovs_momentum: True
        stacking__mlp__power_t: 0.5
        stacking__mlp__random_state: None
        stacking__mlp__shuffle: True
        stacking__mlp__solver: 'adam'
        stacking__mlp__tol: 0.0001
        stacking__mlp__validation_fraction: 0.1
        stacking__mlp__verbose: False
        stacking__mlp__warm_start: False
        stacking__n_jobs: None
        stacking__nb: MultinomialNB()
        stacking__nb__alpha: 1.0
        stacking__nb__class_prior: None
        stacking__nb__fit_prior: True
        stacking__nb__force_alpha: 'warn'
        stacking__passthrough: False
        stacking__stack_method: 'auto'
        stacking__verbose: 0
        steps: [('cv', TfidfVectorizer()), ('stacking',
StackingClassifier(estimators=[('nb', MultinomialNB()),
                              ('mlp', MLPClassifier(alpha=0.1))]))]
        verbose: False

/usr/local/lib/python3.8/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[ ]:
```
#new ensemble
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score



# Define the base estimators for the stacking classifier
ensemble_estimators = [('nb', MultinomialNB()), ('mlp', MLPClassifier())]

ensemble_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    ('stacking', StackingClassifier(estimators=ensemble_estimators))
])

# Define the grid search parameters
ensemble_params = {
    #'cv__max_df': [0.5, 0.75, 1.0],
   # 'nb__alpha': [0.1, 0.5, 1.0],
    "cv__stop_words": [list(stop_words_custom)],
    'stacking__mlp__solver':["lbfgs"],
    'stacking__mlp__hidden_layer_sizes': [(32,)],
   # 'mlp__hidden_layer_sizes': [(50,), (100,), (200,)],
    'stacking__mlp__alpha': [0.1],
}

ensemble_grid = GridSearchCV(ensemble_pipeline, ensemble_params,␣
 ↪cv=5,scoring='accuracy' ,verbose=1, n_jobs=-1)

ensemble_grid.fit(train_x, train_y)

accuracy = round(ensemble_grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {ensemble_grid.best_params_}")

print_best_params(ensemble_grid)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits

/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
```

```
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 94.984.
The winning parameters are {'cv__stop_words': ['i', "i'll", "i'd", "i'm",
"i've", 'ive', 'me', 'myself', 'you', "you'll", "you'd", "you're", "you've",
'yourself', 'he', "he'll", "he'd", "he's", 'him', 'she', "she'll", "she'd",
"she's", 'her', 'it', "it'll", "it'd", "it's", 'itself', 'oneself', 'we',
"we'll", "we'd", "we're", "we've", 'us', 'ourselves', 'they', "they'll",
"they'd", "they're", "they've", 'them', 'themselves', 'everyone', "everyone's",
'everybody', "everybody's", 'someone', "someone's", 'somebody', "somebody's",
'nobody', "nobody's", 'anyone', "anyone's", 'everything', "everything's",
'something', "something's", 'nothing', "nothing's", 'anything', "anything's",
'a', 'an', 'the', 'this', 'that', "that's", 'these', 'those', 'my', 'your',
'yours', 'his', 'hers', 'its', 'our', 'ours', 'own', 'their', 'theirs', 'few',
'much', 'many', 'lot', 'lots', 'some', 'any', 'enough', 'all', 'both', 'half',
'either', 'neither', 'each', 'every', 'certain', 'other', 'another', 'such',
'several', 'multiple', 'rather', 'quite', 'aboard', 'about', 'above', 'across',
'after', 'against', 'along', 'amid', 'amidst', 'among', 'amongst', 'anti',
'around', 'as', 'at', 'away', 'before', 'behind', 'below', 'beneath', 'beside',
'besides', 'between', 'beyond', 'but', 'by', 'concerning', 'considering',
'despite', 'down', 'during', 'except', 'excepting', 'excluding', 'far',
'following', 'for', 'from', 'here', "here's", 'in', 'inside', 'into', 'left',
'like', 'minus', 'near', 'of', 'off', 'on', 'onto', 'opposite', 'out',
'outside', 'over', 'past', 'per', 'plus', 'regarding', 'right', 'since', 'than',
'there', "there's", 'through', 'to', 'toward', 'towards', 'under', 'underneath',
'unlike', 'until', 'up', 'upon', 'versus', 'via', 'with', 'within', 'without',
'may', 'might', 'will', "won't", 'would', "wouldn't", 'can', "can't", 'cannot',
'could', "couldn't", 'should', "shouldn't", 'must', "must've", 'be', 'being',
'been', 'am', 'are', "aren't", "ain't", 'is', "isn't", 'was', "wasn't", 'were',
"weren't", 'do', 'doing', "don't", 'does', "doesn't", 'did', "didn't", 'done',
'have', "haven't", 'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting',
'gets', 'got', 'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make',
'making', 'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need',
'needing', 'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna',
'wanting', 'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing',
'supposes', 'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying',
'says', 'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
```

```
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites'],
'stacking__mlp__alpha': 0.1, 'stacking__mlp__hidden_layer_sizes': (32,),
'stacking__mlp__solver': 'lbfgs'}
        cv: TfidfVectorizer(stop_words=['i', "i'll", "i'd", "i'm", "i've",
'ive', 'me',
                                'myself', 'you', "you'll", "you'd", "you're",
                                "you've", 'yourself', 'he', "he'll", "he'd", "he's",
                                'him', 'she', "she'll", "she'd", "she's", 'her',
                                'it', "it'll", "it'd", "it's", 'itself', 'oneself',
…])
        cv__analyzer: 'word'
        cv__binary: False
        cv__decode_error: 'strict'
        cv__dtype: <class 'numpy.float64'>
        cv__encoding: 'utf-8'
        cv__input: 'content'
        cv__lowercase: True
        cv__max_df: 1.0
```

```
        cv__max_features: None
        cv__min_df: 1
        cv__ngram_range: (1, 1)
        cv__norm: 'l2'
        cv__preprocessor: None
        cv__smooth_idf: True
        cv__stop_words: ['i', "i'll", "i'd", "i'm", "i've", 'ive', 'me',
'myself', 'you', "you'll", "you'd", "you're", "you've", 'yourself', 'he',
"he'll", "he'd", "he's", 'him', 'she', "she'll", "she'd", "she's", 'her', 'it',
"it'll", "it'd", "it's", 'itself', 'oneself', 'we', "we'll", "we'd", "we're",
"we've", 'us', 'ourselves', 'they', "they'll", "they'd", "they're", "they've",
'them', 'themselves', 'everyone', "everyone's", 'everybody', "everybody's",
'someone', "someone's", 'somebody', "somebody's", 'nobody', "nobody's",
'anyone', "anyone's", 'everything', "everything's", 'something', "something's",
'nothing', "nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this',
'that', "that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its',
'our', 'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots',
'some', 'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each',
'every', 'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather',
'quite', 'aboard', 'about', 'above', 'across', 'after', 'against', 'along',
'amid', 'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away',
'before', 'behind', 'below', 'beneath', 'beside', 'besides', 'between',
'beyond', 'but', 'by', 'concerning', 'considering', 'despite', 'down', 'during',
'except', 'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here',
"here's", 'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off',
'on', 'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
```

```
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites']
        cv__strip_accents: None
        cv__sublinear_tf: False
        cv__token_pattern: '(?u)\\b\\w\\w+\\b'
        cv__tokenizer: None
        cv__use_idf: True
        cv__vocabulary: None
        memory: None
        stacking: StackingClassifier(estimators=[('nb', MultinomialNB()),
                                 ('mlp',
                                  MLPClassifier(alpha=0.1,
                                                hidden_layer_sizes=(32,),
                                                solver='lbfgs'))])
        stacking__cv: None
        stacking__estimators: [('nb', MultinomialNB()), ('mlp',
MLPClassifier(alpha=0.1, hidden_layer_sizes=(32,), solver='lbfgs'))]
```

```
        stacking__final_estimator: None
        stacking__mlp: MLPClassifier(alpha=0.1, hidden_layer_sizes=(32,),
solver='lbfgs')
        stacking__mlp__activation: 'relu'
        stacking__mlp__alpha: 0.1
        stacking__mlp__batch_size: 'auto'
        stacking__mlp__beta_1: 0.9
        stacking__mlp__beta_2: 0.999
        stacking__mlp__early_stopping: False
        stacking__mlp__epsilon: 1e-08
        stacking__mlp__hidden_layer_sizes: (32,)
        stacking__mlp__learning_rate: 'constant'
        stacking__mlp__learning_rate_init: 0.001
        stacking__mlp__max_fun: 15000
        stacking__mlp__max_iter: 200
        stacking__mlp__momentum: 0.9
        stacking__mlp__n_iter_no_change: 10
        stacking__mlp__nesterovs_momentum: True
        stacking__mlp__power_t: 0.5
        stacking__mlp__random_state: None
        stacking__mlp__shuffle: True
        stacking__mlp__solver: 'lbfgs'
        stacking__mlp__tol: 0.0001
        stacking__mlp__validation_fraction: 0.1
        stacking__mlp__verbose: False
        stacking__mlp__warm_start: False
        stacking__n_jobs: None
        stacking__nb: MultinomialNB()
        stacking__nb__alpha: 1.0
        stacking__nb__class_prior: None
        stacking__nb__fit_prior: True
        stacking__nb__force_alpha: 'warn'
        stacking__passthrough: False
        stacking__stack_method: 'auto'
        stacking__verbose: 0
        steps: [('cv', TfidfVectorizer(stop_words=['i', "i'll", "i'd", "i'm",
"i've", 'ive', 'me',
                                'myself', 'you', "you'll", "you'd", "you're",
                                "you've", 'yourself', 'he', "he'll", "he'd", "he's",
                                'him', 'she', "she'll", "she'd", "she's", 'her',
                                'it', "it'll", "it'd", "it's", 'itself', 'oneself',
…])), ('stacking', StackingClassifier(estimators=[('nb', MultinomialNB()),
                                   ('mlp',
                                    MLPClassifier(alpha=0.1,
                                                  hidden_layer_sizes=(32,),
                                                  solver='lbfgs'))]))]
        verbose: False
```

[ ]:
```
#new ensemble
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score



# Define the base estimators for the stacking classifier
ensemble_estimators = [('nb', MultinomialNB()), ('mlp', MLPClassifier())]

ensemble_selector = SelectKBest(chi2)
ensemble_normalizer = Normalizer()

ensemble_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    ("normalizer",ensemble_normalizer),
    ("selecter", ensemble_selector),
    ('stacking', StackingClassifier(estimators=ensemble_estimators))
])

# Define the grid search parameters
ensemble_params = {
    'cv__max_df': [0.75],
    "selecter__k":[5000],
    "cv__stop_words": [list(stop_words_custom)],
    "normalizer__norm": ['l2'],
    'cv__ngram_range': [(1,1)],
    'stacking__mlp__solver':["lbfgs"],
    'stacking__mlp__hidden_layer_sizes': [(32,)],
    'stacking__mlp__alpha': [0.1],
    'stacking__nb__alpha': [0.5],
}

ensemble_grid = GridSearchCV(ensemble_pipeline, ensemble_params,␣
 ↪cv=5,scoring='accuracy' ,verbose=1, n_jobs=-1)

ensemble_grid.fit(train_x, train_y)

accuracy = round(ensemble_grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {ensemble_grid.best_params_}")
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits

/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 95.123.
The winning parameters are {'cv__max_df': 0.75, 'cv__ngram_range': (1, 1),
'cv__stop_words': ['i', "i'll", "i'd", "i'm", "i've", 'ive', 'me', 'myself',
'you', "you'll", "you'd", "you're", "you've", 'yourself', 'he', "he'll", "he'd",
"he's", 'him', 'she', "she'll", "she'd", "she's", 'her', 'it', "it'll", "it'd",
"it's", 'itself', 'oneself', 'we', "we'll", "we'd", "we're", "we've", 'us',
'ourselves', 'they', "they'll", "they'd", "they're", "they've", 'them',
'themselves', 'everyone', "everyone's", 'everybody', "everybody's", 'someone',
"someone's", 'somebody', "somebody's", 'nobody', "nobody's", 'anyone',
"anyone's", 'everything', "everything's", 'something', "something's", 'nothing',
"nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this', 'that',
"that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its', 'our',
'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots', 'some',
'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each', 'every',
'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather', 'quite',
'aboard', 'about', 'above', 'across', 'after', 'against', 'along', 'amid',
'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away', 'before',
'behind', 'below', 'beneath', 'beside', 'besides', 'between', 'beyond', 'but',
'by', 'concerning', 'considering', 'despite', 'down', 'during', 'except',
'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here', "here's",
'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off', 'on',
'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
```

```
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites'], 'normalizer__norm':
'l2', 'selecter__k': 5000, 'stacking__mlp__alpha': 0.1,
'stacking__mlp__hidden_layer_sizes': (32,), 'stacking__mlp__solver': 'lbfgs',
'stacking__nb__alpha': 0.5}
```

[ ]:
```
#new ensemble
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score



# Define the base estimators for the stacking classifier
ensemble_estimators = [('nb', MultinomialNB()), ('mlp', MLPClassifier())]

ensemble_selector = SelectKBest(chi2)
ensemble_normalizer = Normalizer()

ensemble_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    ("normalizer",ensemble_normalizer),
    ("selecter", ensemble_selector),
    ('stacking', StackingClassifier(estimators=ensemble_estimators))
])

# Define the grid search parameters
ensemble_params = {
    'cv__max_df': [0.75],
    "selecter__k":[5000],
    "cv__stop_words": [list(stop_words_library)],
    "normalizer__norm": ['l2'],
    'cv__ngram_range': [(1,1)],
    'stacking__mlp__solver':["lbfgs"],
    'stacking__mlp__hidden_layer_sizes': [(32,)],
    'stacking__mlp__alpha': [0.1],
    'stacking__nb__alpha': [0.5],
}

ensemble_grid = GridSearchCV(ensemble_pipeline, ensemble_params,␣
 ↪cv=5,scoring='accuracy' ,verbose=1, n_jobs=-1)

ensemble_grid.fit(train_x, train_y)

accuracy = round(ensemble_grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {ensemble_grid.best_params_}")
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
The best accuracy is 95.123.
The winning parameters are {'cv__max_df': 0.75, 'cv__ngram_range': (1, 1),
'cv__stop_words': ['even', 'put', 'eleven', 'won', 'didn', 'beforehand',
'toward', 'couldnt', 'mostly', 'eight', 'either', 'enough', 'your', 'while',
'been', 'anyway', 'sincere', 'hasnt', 'others', 'another', 'none', 'itself',
```

```
'as', 'formerly', 'often', 'about', 'off', 'just', 'during', 't', 'cannot',
'rather', "aren't", 'too', 'ever', 'wasn', 'less', 'yourself', 'myself', 'do',
'hereafter', "that'll", 'became', 'will', 'back', "haven't", 'seemed', 'name',
'one', 'never', 'so', 'onto', "wasn't", 'find', 'until', 'if', "won't", 'here',
'elsewhere', 'no', 'those', 'needn', 'hence', 'meanwhile', 'from', 'hereupon',
'for', 'almost', 'did', 'least', 'with', 'she', 'many', 'without', 'noone',
'thereupon', 'not', 'my', 'throughout', 'thick', 'such', 'hadn', 'us', 'all',
'now', 'twenty', 'once', 'at', 'fifty', 'anywhere', 'whereas', 'former', 'else',
'always', 'sometimes', 'please', "mightn't", 'mightn', 'd', 'same', 'other',
'few', 'nobody', 'describe', 'sometime', 'somewhere', 'etc', 'seem', 'seems',
"needn't", 'mill', 'which', 'thereafter', 'sixty', 'together', 'therein', 'two',
"shan't", 'between', 'he', 'thin', 'already', 'his', 'their', 'hereby', 'doing',
'indeed', 'first', 'latterly', 'still', 'or', 'm', 'nor', 'can', 'neither',
"hasn't", 'll', 'next', 'when', 'thru', 'over', 'hers', 'mustn', 'besides',
'could', 'side', 'ten', 'yourselves', 'move', 'nevertheless', 'ours', 'this',
'perhaps', 'fifteen', "it's", 'well', 'con', 'up', 'un', 'be', 'mine', 'around',
'has', 'whatever', 'wouldn', 'them', 'five', 'last', 'each', "you're",
'nowhere', 'shouldn', 'wherever', 'ie', 'anyone', 'again', 'were', 'via',
'theirs', 'being', 'anyhow', 'it', 'more', 'under', 'have', 'since', 'through',
'having', "you'll", 'four', 'whereby', 'anything', 'front', 'afterwards', 'a',
'does', 's', 'six', 'somehow', 'should', 'shan', 'would', 'its', 'isn', 'any',
'where', 'keep', 'per', 'also', 'among', 'only', 'except', 'must', 'though',
'take', 'amoungst', 'behind', "isn't", 'of', 'done', 'show', 'own', 'by',
"shouldn't", "weren't", 'give', 'after', 'twelve', "don't", 'thence',
"wouldn't", "you've", 'then', 'these', 'to', 'everything', 'namely', "you'd",
'beside', 'i', 'ltd', 'don', 'me', 'due', "hadn't", 'hasn', 'made', 'whoever',
'above', 'forty', 'themselves', 'both', 'hundred', 're', 'our', 'amongst',
'however', 'moreover', 'out', 'fill', "couldn't", 'down', 'whom', 'become',
'haven', 'weren', 'thus', 'ma', 'below', 'becomes', 'everywhere', 'interest',
'much', 'herein', 'yours', 'seeming', 'is', 'nine', 'full', 'ourselves', 'ain',
'latter', 'across', 'am', 'call', 'whereupon', 'something', "doesn't", 'found',
'why', 'most', 'therefore', 'co', 'thereby', 'someone', 'empty', 'on', 'who',
'towards', 'whereafter', 'go', 'there', 'cry', 'they', 'because', 'beyond',
'bottom', 'that', 'de', 'further', 'y', 'very', 'whole', 'get', 'alone', 'than',
'detail', 'and', 'part', 'whenever', 'top', 'every', 'him', 'but', 'amount',
'everyone', 'herself', 'aren', 'along', 'three', 'fire', 'against', 'we',
"she's", 'becoming', 've', 'are', 'bill', 'before', "mustn't", 'within',
'wherein', 'doesn', 'was', 'nothing', 'himself', 'the', 'whence', 'whither',
'otherwise', 'serious', 'eg', 'in', 'inc', 'into', 'o', 'some', 'upon',
'whether', 'yet', 'cant', 'several', 'how', 'had', 'may', 'whose', "should've",
'system', "didn't", 'an', 'third', 'her', 'see', 'couldn', 'although', 'you',
'might', 'what'], 'normalizer__norm': 'l2', 'selecter__k': 5000,
'stacking__mlp__alpha': 0.1, 'stacking__mlp__hidden_layer_sizes': (32,),
'stacking__mlp__solver': 'lbfgs', 'stacking__nb__alpha': 0.5}
```

[ ]:
```
#new ensemble
from sklearn.datasets import load_digits
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score



# Define the base estimators for the stacking classifier
ensemble_estimators = [('nb', MultinomialNB()), ('mlp', MLPClassifier())]

ensemble_selector = SelectKBest(chi2)
ensemble_normalizer = Normalizer()

ensemble_pipeline = Pipeline([
    ('cv', TfidfVectorizer()),
    ("normalizer",ensemble_normalizer),
    ("selecter", ensemble_selector),
    ('stacking', StackingClassifier(estimators=ensemble_estimators))
])

# Define the grid search parameters
ensemble_params = {
    'cv__max_df': [1.0],
    "selecter__k":[5000],
    "cv__stop_words": [list(stop_words_custom)],
    'cv__preprocessor': [preprocess_text],
    #    'cv__preprocessor': [preprocess_text,remove_punctuation,None],

    "cv__binary": [False],
    "normalizer__norm": ['l2'],
    'cv__ngram_range': [(1,1)],
    'stacking__mlp__solver':["lbfgs"],
    'stacking__mlp__hidden_layer_sizes': [(32,)],
    'stacking__mlp__alpha': [0.1],
    'stacking__nb__alpha': [0.1],
}

ensemble_grid = GridSearchCV(ensemble_pipeline, ensemble_params,␣
 ↪cv=5,scoring='accuracy' ,verbose=1, n_jobs=-1)

ensemble_grid.fit(train_x, train_y)

accuracy = round(ensemble_grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
```

```
print(f"The winning parameters are {ensemble_grid.best_params_}")
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits

/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:409:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ['ain', 'aren', 'couldn', 'didn',
'doesn', 'hadn', 'hasn', 'haven', 'isn', 'll', 're', 'shouldn', 'site', 'sites',
've', 'wasn', 'web', 'weren', 'won', 'wouldn'] not in stop_words.
  warnings.warn(

The best accuracy is 95.402.
The winning parameters are {'cv__binary': False, 'cv__max_df': 1.0,
'cv__ngram_range': (1, 1), 'cv__preprocessor': <function preprocess_text at
0x7f6cc558ea60>, 'cv__stop_words': ['i', "i'll", "i'd", "i'm", "i've", 'ive',
'me', 'myself', 'you', "you'll", "you'd", "you're", "you've", 'yourself', 'he',
"he'll", "he'd", "he's", 'him', 'she', "she'll", "she'd", "she's", 'her', 'it',
"it'll", "it'd", "it's", 'itself', 'oneself', 'we', "we'll", "we'd", "we're",
"we've", 'us', 'ourselves', 'they', "they'll", "they'd", "they're", "they've",
'them', 'themselves', 'everyone', "everyone's", 'everybody', "everybody's",
'someone', "someone's", 'somebody', "somebody's", 'nobody', "nobody's",
'anyone', "anyone's", 'everything', "everything's", 'something', "something's",
'nothing', "nothing's", 'anything', "anything's", 'a', 'an', 'the', 'this',
'that', "that's", 'these', 'those', 'my', 'your', 'yours', 'his', 'hers', 'its',
'our', 'ours', 'own', 'their', 'theirs', 'few', 'much', 'many', 'lot', 'lots',
'some', 'any', 'enough', 'all', 'both', 'half', 'either', 'neither', 'each',
'every', 'certain', 'other', 'another', 'such', 'several', 'multiple', 'rather',
'quite', 'aboard', 'about', 'above', 'across', 'after', 'against', 'along',
'amid', 'amidst', 'among', 'amongst', 'anti', 'around', 'as', 'at', 'away',
'before', 'behind', 'below', 'beneath', 'beside', 'besides', 'between',
'beyond', 'but', 'by', 'concerning', 'considering', 'despite', 'down', 'during',
'except', 'excepting', 'excluding', 'far', 'following', 'for', 'from', 'here',
"here's", 'in', 'inside', 'into', 'left', 'like', 'minus', 'near', 'of', 'off',
'on', 'onto', 'opposite', 'out', 'outside', 'over', 'past', 'per', 'plus',
'regarding', 'right', 'since', 'than', 'there', "there's", 'through', 'to',
'toward', 'towards', 'under', 'underneath', 'unlike', 'until', 'up', 'upon',
'versus', 'via', 'with', 'within', 'without', 'may', 'might', 'will', "won't",
'would', "wouldn't", 'can', "can't", 'cannot', 'could', "couldn't", 'should',
"shouldn't", 'must', "must've", 'be', 'being', 'been', 'am', 'are', "aren't",
"ain't", 'is', "isn't", 'was', "wasn't", 'were', "weren't", 'do', 'doing',
"don't", 'does', "doesn't", 'did', "didn't", 'done', 'have', "haven't",
'having', 'has', "hasn't", 'had', "hadn't", 'get', 'getting', 'gets', 'got',
'gotten', 'go', 'going', 'gonna', 'goes', 'went', 'gone', 'make', 'making',
'makes', 'made', 'take', 'taking', 'takes', 'took', 'taken', 'need', 'needing',
'needs', 'needed', 'use', 'using', 'uses', 'used', 'want', 'wanna', 'wanting',
'wants', 'let', 'lets', 'letting', "let's", 'suppose', 'supposing', 'supposes',
'supposed', 'seem', 'seeming', 'seems', 'seemed', 'say', 'saying', 'says',
'said', 'know', 'knowing', 'knows', 'knew', 'known', 'look', 'looking',
```

```
'looked', 'think', 'thinking', 'thinks', 'thought', 'feel', 'feels', 'felt',
'based', 'put', 'puts', 'who', "who's", "who've", "who'd", 'whoever',
"whoever's", 'whom', 'whomever', "whomever's", 'whose', 'whosever',
"whosever's", 'when', 'whenever', 'which', 'whichever', 'where', "where's",
"where'd", 'wherever', 'why', "why's", "why'd", 'whyever', 'what', "what's",
'whatever', 'whence', 'how', "how's", "how'd", 'however', 'whether',
'whatsoever', 'and', 'or', 'not', 'because', 'also', 'always', 'never', 'only',
'really', 'very', 'greatly', 'extremely', 'somewhat', 'no', 'nope', 'nah',
'yes', 'yep', 'yeh', 'yeah', 'maybe', 'perhaps', 'more', 'most', 'less',
'least', 'good', 'great', 'well', 'better', 'best', 'bad', 'worse', 'worst',
'too', 'thru', 'though', 'although', 'yet', 'already', 'then', 'even', 'now',
'sometimes', 'still', 'together', 'altogether', 'entirely', 'fully', 'entire',
'whole', 'completely', 'utterly', 'seemingly', 'apparently', 'clearly',
'obviously', 'actually', 'actual', 'usually', 'usual', 'literally', 'honestly',
'absolutely', 'definitely', 'generally', 'totally', 'finally', 'basically',
'essentially', 'fundamentally', 'automatically', 'immediately', 'necessarily',
'primarily', 'normally', 'perfectly', 'constantly', 'particularly',
'eventually', 'hopefully', 'mainly', 'typically', 'specifically', 'differently',
'appropriately', 'plenty', 'certainly', 'unfortunately', 'ultimately',
'unlikely', 'likely', 'potentially', 'fortunately', 'personally', 'directly',
'indirectly', 'nearly', 'closely', 'slightly', 'probably', 'possibly',
'especially', 'frequently', 'often', 'oftentimes', 'seldom', 'rarely', 'sure',
'while', 'whilst', 'able', 'unable', 'else', 'ever', 'once', 'twice', 'thrice',
'almost', 'again', 'instead', 'next', 'previous', 'unless', 'somehow', 'anyhow',
'anywhere', 'somewhere', 'everywhere', 'nowhere', 'further', 'anymore', 'later',
'ago', 'ahead', 'just', 'same', 'different', 'big', 'small', 'little', 'tiny',
'large', 'huge', 'pretty', 'mostly', 'anyway', 'anyways', 'otherwise',
'regardless', 'throughout', 'additionally', 'moreover', 'furthermore',
'meanwhile', 'afterwards', 'thing', "thing's", 'things', 'stuff', "other's",
'others', "another's", 'total', '', 'false', 'none', 'way', 'kind', 'zero',
'zeros', 'zeroes', 'one', 'ones', 'two', 'three', 'four', 'five', 'six',
'seven', 'eight', 'nine', 'ten', 'twenty', 'thirty', 'forty', 'fifty', 'sixty',
'seventy', 'eighty', 'ninety', 'hundred', 'hundreds', 'thousand', 'thousands',
'million', 'millions', 'first', 'last', 'second', 'third', 'fourth', 'fifth',
'sixth', 'seventh', 'eigth', 'ninth', 'tenth', 'firstly', 'secondly', 'thirdly',
'lastly', 'hello', 'hi', 'hey', 'sup', 'yo', 'greetings', 'please', 'okay',
'ok', "y'all", 'lol', 'rofl', 'thank', 'thanks', 'alright', 'kinda', 'dont',
'sorry', 'idk', 'tldr', 'tl', 'dr', 'tbh', 'dude', 'tho', 'aka', 'plz', 'pls',
'bit', 'don', 'www', 'https', 'http', 'com', 'etchtml', 'reddit', 'subreddit',
'subreddits', 'comments', 'reply', 'replies', 'thread', 'threads', 'post',
'posts', 'website', 'websites', 'web site', 'web sites'], 'normalizer__norm':
'l2', 'selecter__k': 5000, 'stacking__mlp__alpha': 0.1,
'stacking__mlp__hidden_layer_sizes': (32,), 'stacking__mlp__solver': 'lbfgs',
'stacking__nb__alpha': 0.1}
```

[ ]:
```
print(f"The best accuracy is {accuracy}.")
```

```
The best accuracy is 95.402.
```

[ ]:
```
y_pred_new = ensemble_grid.predict(test_x)
create_test_csv(y_pred_new,"Stacking_MultiNB-MLP-05032023_02.csv")
```

```
File saved.
```

[ ]:
```
############################################################ final
```