

---

# Predicting Reddit Posts Categories Using Multiclass Naïve Bayes and Random Forest

---

**Max Ardito**

maxwell.ardito@mail.mcgill.ca

**Edwin Meriaux**

edwin.meriaux@mail.mcgill.ca

**Ohood Sabr**

ohood.sabr@mail.mcgill.ca

## Abstract

One of the most common machine learning challenges is text classification. In this project, a classifier is developed that can predict posts and comments from the site Reddit.com as belonging to one of four subreddits. A multinomial Naive Bayes classifier and a random forest are two independent algorithms that are employed and experimented with. The presented approach heavily utilizes feature construction to produce an ideal vocabulary for the task at hand. The multinomial Naive Bayes classifier was seen to produce the best results under 10-fold cross-validation. An additional test set was then run through this classifier on Kaggle resulting in a preliminary accuracy of 77.8%.

## 1 Introduction

The need to classify and arrange the increasing amount of electronic documents being produced around the world makes text classification one of the most important topics in the field of machine learning. Text classification models have been successfully used for a number of different scenarios, including topic detection, spam e-mail filtering, author identification, and so on [7]. The challenge presented in this paper is to classify unlabeled Reddit posts to their corresponding "Subreddits." Reddit is a social media platform and discussion forum based in the United States in which users can post text, images, links, and videos to one of many different subreddits—message boards that group posts around a central topic. The unlabeled Reddit posts in question each belong to one of four possible subreddits: JavaScript, MATLAB, PyTorch, and TensorFlow.

We have used two methods to solve this classification issue. The first method is a multinomial Naive Bayes (MNB) classifier which assumes that the features are completely independent of each other. The second is a random forest method in which various decision trees are trained in parallel using common bagging and bootstrapping methods. According to the mean test accuracy in 10-fold cross-validation, the Naive Bayes classifier produced the best results after both carefully preprocessing the compiled training set of Reddit posts into a vectorized vocabulary and choosing the optimal hyperparameters. Laplace smoothing was used with the classifier to address the issue of zero probability. Additionally, a number of variations on the model's input vocabulary were evaluated over the course of our experimentation.

This report is organized as follows: Section 2 discusses the data set and talks about the preprocessing steps that were taken. Section 3 presents the various models experimented with during the training phase. Section 4 presents the findings, and Section 5 of the report concludes with some possible further research.

## 2 Datasets

### 2.1 Overview

The provided training and testing sets for evaluating our model were collected directly from the aforementioned subreddits. The training dataset has 720 samples which represent posts and comments that belong to the JavaScript, MATLAB, PyTorch, and TensorFlow subreddits. Likewise, the test set consists of 380 posts and comments. The classes are distributed equally in the training dataset. In order to construct the most relevant aspects of the input data, individual words will be treated as features. Furthermore, these words will undergo a number of preprocessing steps to filter out irrelevant features.

### 2.2 Features Extraction

Perhaps the most obvious issue with this classification task is the inherent crossover regarding vocabulary used in posts from these four subreddits. Since the four subreddits all deal with programming, care must be taken not only in constructing a dictionary of common stop words, but also in constructing a potential list of extended stop words that are common to all four programming languages, e.g. *function*, *package*, *variable*, etc. At the very least, the overlapping vocabulary must be weighted differently than vocabulary that is unique to one of the four languages, for instance *npm* for JavaScript, or *keras* for TensorFlow.

To first clear our vocabulary of common words, we considered a 318-word standard English stop-word dictionary from the Natural Language Toolkit Python module.[6] We then created our own list of additional English language stop words and merged it with the previous stop-word dictionary. Our next task was to remove all punctuation, capitalization, and spaces in order to obtain lists of lowercase words. This is a crucial step, as many of the posts in the training set consist of code blocks containing characters that are common to most programming languages such as curly braces , parenthesis ( ), and backslashes //. Finally, Lemmatization was applied on the training set to eliminate grammatical variations from the vocabulary by extract a common root from each term.

### 2.3 Feature Analysis

After cleaning both the training and testing dataset, a vectorization process is performed in order to obtain a final vocabulary of features for our model. SciKit-Learn’s CountVectorizer was used to vectorize, lemmatize, and filter out the provided dictionary of stop-words. Thus, the output of this process is a vector with each dimension representing a word in the final word list. This vector can be used to analyze the frequency with which each word in the vector appears in a given input post. While training the MNB model (see Section 3.2), it was seen that an increase in accuracy was achieved by integrating 2-grams and 3-grams into the original feature space. This accuracy increase is obtained at the cost of efficiency, as the resulting size of the feature space must increase by at least a factor of 3.

The TF-IDF technique is frequently used to rate each word in a text document according to how unique it is. In other words, the TF-IDF technique captures the relevance between words, text documents, and certain categories [8] . It aims to assess a word’s significance to a document inside a corpus (or collection) of documents by assigning a score or weight to each word. Along with the aforementioned preprocessing techniques, TF-IDF was implemented in our feature vector using the SciKit Learn’s TfidfTransformer[1].

$$IDF(t, Corpus) = \log_2 \frac{(\#Docs\ in\ Corpus)}{(\#Docs\ with\ term\ t) + 1} \quad (1)$$



Figure 1: Histograms of representing the frequency of occurrence of each word from the post-processed vocabulary in training set’s JavaScript, MATLAB, PyTorch, and TensorFlow posts respectively. Words are organized in descending order according to total number of occurrences in the input data.

Table 1: Top Features and Scores Using Chi-Squared Metric

tensorflow	matlab	pytorch	tf	mathworks	torch	js	keras	model	function
30.28	28.61	21.75	18.96	16.61	15.60	13.08	11.24	11.09	9.37

In Figure 1, the multiclass overlap between the 20 most frequently words is shown in the form of a histogram. In this study, the chi-squared statistic is employed as an analysis tool on our final feature space, as it is a suitable metric for calculating the relevance of certain words in the document corpus[5]. Additionally, Table 1 shows the most important features along with their scores.

### 3 Proposed Approach

After taking the necessary steps to preprocess and vectorize our feature space, an optimal model must be selected by means of experimentation and evaluation. A multinomial Naïve Bayes classifier was chosen as a base model due to its common use in text classification tasks [3]. Using 10-fold cross validation, the MNB model was evaluated multiple times to confirm and verify the success of the various preprocessing steps that were described in Section 2.3. Hyper-parameters such as N-gram ranges, maximum number of features, and more refined stop-word filtering were experimented with during this cross validation step.

An additional model was also tested which utilized a random forest approach, consisting of bootstrapping the training using  $N$  different decision trees and bagging their results. The performance of the random forest approach was compared with the MNB result by once again using 10-fold cross validation.

### 3.1 Multinomial Naive Bayes

The vectorized data obtained from Section 2 can be used to train a MNB model using a few different methods. For this task, we opted to implement a one-against-all approach, which treats a four-class classification problem as four separate binary classification problems. New posts are thus predicted by running them through each of the four classifiers and choosing the highest resulting sigma value.

The perks of this approach are that it results in conceptual simplicity and easily readable code. However this is at the expense of scalability, since four separate models must be trained. Because the provided training set is relatively small, the authors decided to go ahead with this approach.

In order to classify new data using this MNB model, we first must train the model to obtain parameters for each of the four classifiers. These include...

1.  $\theta_n$ : The total number of training samples where  $y$  belongs to subreddit  $n$  over the total number of training samples
2.  $\theta_{j,n}$ : The total number of training samples where word  $j$  in the vectorized vocabulary  $x$  exists in a post that also belongs to subreddit  $n$  over the total number of posts belonging to subreddit  $n$
3.  $\theta_{j,\neg n}$ : The total number of training samples where word  $j$  in the vectorized vocabulary  $x$  exists in a post that does not belong to subreddit  $n$  over the total number of posts that do not belong to subreddit  $n$

Once these values are obtained, a new input post  $\hat{x}$  can be vectorized and evaluated using the log-odds decision boundary formulated in (2). This equation is a slightly modified version of the equation found in [7].

$$a_n(\hat{x}) = \log \left( \frac{\theta_n}{1 - \theta_n} \right) + \sum_{j=0}^m \left( x_j \log \frac{\theta_{j,n}}{\theta_{j,\neg n}} + (1 - x_j) \log \frac{(1 - \theta_{j,n})}{(1 - \theta_{j,\neg n})} \right) \quad (2)$$

The resulting class for  $\hat{x}$  can thus be found by evaluating each class' log-odds ratio and finding  $\max_n \{ \sigma(a_n(x)) \} \forall n$ , where  $\sigma$  is the sigmoid function.

### 3.2 Random Forest

A random forest base model was also experimented with on the dataset. The process of building a random forest model starts with constructing a set of  $B$  different decision trees that each contain  $m'$  features, where  $m' < m$  and  $m$  is the total number of features in the feature space [2]. These decision trees are organized according to each node's information gain  $IG(Y|X)$ .

The random forest algorithm is different from a regular decision tree in that it bags multiple different decision trees together to produce a mean result. Bagging refers to the process of using the same algorithm in  $n$  instances, which are each trained on bootstrapped fractions of the data set. The results of the  $n$  different instances are evaluated using either a majority vote for classification or a mean average for regression. This results in a model that combats overfitting at the expense of computational complexity [4]. Bagging compares to stacking as a somewhat similar methods. Stacking is like bagging but with different algorithms rather than just differently trained versions of the same algorithms. These algorithms can be trained on different sets of the algorithm like in bagging.

## 4 Results

Between the random forest model and the MNB model, the classifier that scored the highest mean accuracy on 10-fold cross validation was the MNB model with the aforementioned vocabulary using lemmatization, TF-IDF, 2-grams, and 3-grams. The optimal number of features used in vectorization was seen to be the first 1000 using the `max_features` argument in SciKit Learn's

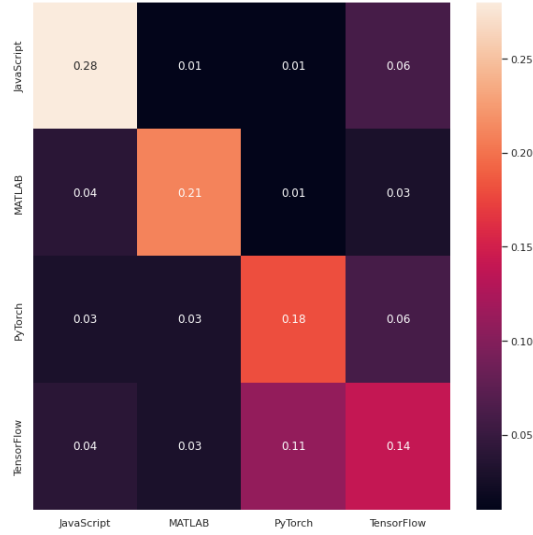


Figure 2: Multiclass confusion matrix showing the performance of the base MNB model in terms of the predicted classes on the Y-axis and the actual classes on the X-axis.

CountVectorizer. This model scored 77.18%. The random forest model occasionally performed better and occasionally performed worse, scoring a margin of 2% higher or lower than the MNB.

A subsequent improvement was attempted on the model pertaining to the differences between the PyTorch and TensorFlow vocabulary. These two subreddits contain vocabularies that are relatively similar to each other, since they both consist of Python-related topics and questions. This can be seen in Figure 2, which shows that the bulk of the model’s misclassifications occurred in classifying a PyTorch post as a TensorFlow post and vice versa. In an attempt to resolve this issue, the terms *PyTorch*, *TensorFlow*, *torch*, *keras*, *tf*, and *tensor* were manually reweighted in our MNB model. Weights belonging to these elements of the feature space were increased by a factor of 100, resulting in an increased accuracy of 78.31% on the cross validation tests.

A separate testing set was also used on Kaggle.com to test the model when trained on the entirety of its training data. The model that scored the most accuracy on the Kaggle test set was the MNB model, which performed with little difference to the cross-validation tests and scored 77.8% accuracy, displaying its durability regarding variance.

## 5 Discussion and conclusion

In our final results it was found that the MNB model not only performed best but also held up well regarding overfitting. This can be seen in the similarity between the mean cross-validation score and the test set score. However, this overall accuracy of the model could certainly be improved. Given an even more attentive preprocessing, a larger dataset, and a slightly more complex model, it is possible that the accuracy could improve without sacrificing its low variance.

## 6 Statement of Contribution

1. Max Ardito: Naive Bayes base model, plots, feature analysis, Pytorch/TF reweighting, report.
2. Ohood Sabr: Feature construction, feature analysis, lemmatization, Chi Squared analysis, TF-IDF, report.
3. Edwin Meriaux: Feature construction, word and symbol filtering and separation, stop words bagging, stacking, random forest, report.

## References

- [1] Narges Armanfard. Ecse 551 - machine learning for engineers lecture 12 — decision trees (cont'd), feature construction, dimension reduction, September 2022.
- [2] Narges Armanfard. Ecse 551 - machine learning for engineers lecture 17 — ensemble methods, September 2022.
- [3] Narges Armanfard. Ecse 551 - machine learning for engineers lecture 9 — naive bayes, September 2022.
- [4] IBM Cloud Education. Random forest.
- [5] Dr. Saptarsi Goswami. Using the chi-squared test for feature selection with implementation, 2020.
- [6] omdena.com/blog/machine-learning-classification-algorithms. Text pre-processing: Stop words removal using different libraries, February.
- [7] Alper Kursat Uysal and Serkan Gunal. The impact of preprocessing on text classification. *Information Processing Management*, 50(1):104–112, 2014.
- [8] Zhang Yun-tao, Gong Ling, and Wang Yong-cheng. An improved tf-idf approach for text classification. *Journal of Zhejiang University-Science A*, 6(1):49–55, 2005.

## 7 Appendix A: Code

```
1  # -*- coding: utf-8 -*-
2  """Reddit_NLP_Predictor_(Final).ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/16ZPRnjjbSfHpP472HiaP-xbSIN5JIm5a
8
9  # Subreddit Predictor
10
11  Implementation of a model that predicts whether input posts are from 1 of 4
12  ↳ different programming subreddits. Uses a multiclass Bernoulli Naive Bayes
13  ↳ model with stop words and bagging
14
15  ## Install (Most) Dependencies
16  """
17
18  # Installing natural language toolkit
19  !pip install nltk
20  import nltk
21  nltk.download('punkt')
22  nltk.download('wordnet')
23  nltk.download('averaged_perceptron_tagger')
24
25  #import list
26  import numpy as np
27  import random as rand
28  import pandas as pd
29  import seaborn as sns
30  import string
31  from scipy import sparse
32  from sklearn.ensemble import BaggingClassifier
33  from sklearn.ensemble import AdaBoostClassifier
34  from sklearn.model_selection import KFold
35  from sklearn.tree import DecisionTreeClassifier
36  from sklearn.naive_bayes import GaussianNB, MultinomialNB
37  from sklearn.linear_model import LogisticRegression
```

```

36 from sklearn.svm import LinearSVC
37 from sklearn.linear_model import SGDClassifier
38 from sklearn.model_selection import train_test_split
39 from sklearn.feature_extraction.text import CountVectorizer
40 from sklearn.feature_extraction import text
41 from sklearn.feature_extraction.text import TfidfTransformer
42 from sklearn.feature_extraction.text import TfidfVectorizer
43 from sklearn.preprocessing import Normalizer
44 from nltk.corpus import wordnet
45 from nltk import word_tokenize
46 from nltk.stem import WordNetLemmatizer
47 from nltk.stem import PorterStemmer
48 from sklearn.pipeline import Pipeline
49 from sklearn.model_selection import GridSearchCV
50 from sklearn.feature_selection import SelectKBest, chi2, f_classif,
↳ mutual_info_classif, f_regression, mutual_info_regression,
↳ SelectPercentile
51 import math as ma
52 import scipy as sp
53 import matplotlib.pyplot as plt
54 import pandas as pd
55 import time
56 print("Finished importing!")
57
58 """## Load The Dataset"""
59
60 import pandas as pd
61 import requests
62 from io import StringIO
63 import io
64 from google.colab import files
65
66 #imports train set from google drive
67 url='https://drive.google.com/file/d/1cqbyoylVGKr9LZ_5PhJFhvkbw8sCH2/view?usp=sharing'
68
69 file_id = url.split('/')[2]
70 dwn_url='https://drive.google.com/uc?export=download&id=' + file_id
71 url2 = requests.get(dwn_url).text
72 csv_raw = StringIO(url2)
73
74
75 df = pd.read_csv(csv_raw)
76 df.head()
77
78 #imports test set from computer local
79 uploaded = files.upload()
80 df2 = pd.read_csv(io.BytesIO(uploaded['test.csv']), encoding='cp1252')
81 print(df2.iloc[:,1].to_numpy())
82 test_set = df2.iloc[:,1].to_numpy()
83
84 """## Custom Stop Words"""
85
86 #custom word list to improve accuracy to filter out specific words
87 #partially duplicated in the imported words list
88 switcher = {
89     # All pronouns and associated words
90     "i": True,
91     "i'll": True,
92     "i'd": True,
93     "i'm": True,
94     "i've": True,
95     "ive": True,
96     "me": True,
97     "myself": True,

```

```

98 "you": True,
99 "you'll": True,
100 "you'd": True,
101 "you're": True,
102 "you've": True,
103 "yourself": True,
104 "he": True,
105 "he'll": True,
106 "he'd": True,
107 "he's": True,
108 "him": True,
109 "she": True,
110 "she'll": True,
111 "she'd": True,
112 "she's": True,
113 "her": True,
114 "it": True,
115 "it'll": True,
116 "it'd": True,
117 "it's": True,
118 "itself": True,
119 "oneself": True,
120 "we": True,
121 "we'll": True,
122 "we'd": True,
123 "we're": True,
124 "we've": True,
125 "us": True,
126 "ourselves": True,
127 "they": True,
128 "they'll": True,
129 "they'd": True,
130 "they're": True,
131 "they've": True,
132 "them": True,
133 "themselves": True,
134 "everyone": True,
135 "everyone's": True,
136 "everybody": True,
137 "everybody's": True,
138 "someone": True,
139 "someone's": True,
140 "somebody": True,
141 "somebody's": True,
142 "nobody": True,
143 "nobody's": True,
144 "anyone": True,
145 "anyone's": True,
146 "everything": True,
147 "everything's": True,
148 "something": True,
149 "something's": True,
150 "nothing": True,
151 "nothing's": True,
152 "anything": True,
153 "anything's": True,
154 # All determiners and associated words
155 "a": True,
156 "an": True,
157 "the": True,
158 "this": True,
159 "that": True,
160 "that's": True,
161 "these": True,

```



```

162 "those": True,
163 "my": True,
164 "your": True,
165 "yours": True,
166 "his": True,
167 "hers": True,
168 "its": True,
169 "our": True,
170 "ours": True,
171 "own": True,
172 "their": True,
173 "theirs": True,
174 "few": True,
175 "much": True,
176 "many": True,
177 "lot": True,
178 "lots": True,
179 "some": True,
180 "any": True,
181 "enough": True,
182 "all": True,
183 "both": True,
184 "half": True,
185 "either": True,
186 "neither": True,
187 "each": True,
188 "every": True,
189 "certain": True,
190 "other": True,
191 "another": True,
192 "such": True,
193 "several": True,
194 "multiple": True,
195 # "what": True,      #Dealt with later on
196 "rather": True,
197 "quite": True,
198 # All prepositions
199 "aboard": True,
200 "about": True,
201 "above": True,
202 "across": True,
203 "after": True,
204 "against": True,
205 "along": True,
206 "amid": True,
207 "amidst": True,
208 "among": True,
209 "amongst": True,
210 "anti": True,
211 "around": True,
212 "as": True,
213 "at": True,
214 "away": True,
215 "before": True,
216 "behind": True,
217 "below": True,
218 "beneath": True,
219 "beside": True,
220 "besides": True,
221 "between": True,
222 "beyond": True,
223 "but": True,
224 "by": True,
225 "concerning": True,

```

```

226 "considering": True,
227 "despite": True,
228 "down": True,
229 "during": True,
230 "except": True,
231 "excepting": True,
232 "excluding": True,
233 "far": True,
234 "following": True,
235 "for": True,
236 "from": True,
237 "here": True,
238 "here's": True,
239 "in": True,
240 "inside": True,
241 "into": True,
242 "left": True,
243 "like": True,
244 "minus": True,
245 "near": True,
246 "of": True,
247 "off": True,
248 "on": True,
249 "onto": True,
250 "opposite": True,
251 "out": True,
252 "outside": True,
253 "over": True,
254 "past": True,
255 "per": True,
256 "plus": True,
257 "regarding": True,
258 "right": True,
259 "since": True,
260 "than": True,
261 "there": True,
262 "there's": True,
263 "through": True,
264 "to": True,
265 "toward": True,
266 "towards": True,
267 "under": True,
268 "underneath": True,
269 "unlike": True,
270 "until": True,
271 "up": True,
272 "upon": True,
273 "versus": True,
274 "via": True,
275 "with": True,
276 "within": True,
277 "without": True,
278 # Irrelevant verbs
279 "may": True,
280 "might": True,
281 "will": True,
282 "won't": True,
283 "would": True,
284 "wouldn't": True,
285 "can": True,
286 "can't": True,
287 "cannot": True,
288 "could": True,
289 "couldn't": True,

```

```
290 "should": True,
291 "shouldn't": True,
292 "must": True,
293 "must've": True,
294 "be": True,
295 "being": True,
296 "been": True,
297 "am": True,
298 "are": True,
299 "aren't": True,
300 "ain't": True,
301 "is": True,
302 "isn't": True,
303 "was": True,
304 "wasn't": True,
305 "were": True,
306 "weren't": True,
307 "do": True,
308 "doing": True,
309 "don't": True,
310 "does": True,
311 "doesn't": True,
312 "did": True,
313 "didn't": True,
314 "done": True,
315 "have": True,
316 "haven't": True,
317 "having": True,
318 "has": True,
319 "hasn't": True,
320 "had": True,
321 "hadn't": True,
322 "get": True,
323 "getting": True,
324 "gets": True,
325 "got": True,
326 "gotten": True,
327 "go": True,
328 "going": True,
329 "gonna": True,
330 "goes": True,
331 "went": True,
332 "gone": True,
333 "make": True,
334 "making": True,
335 "makes": True,
336 "made": True,
337 "take": True,
338 "taking": True,
339 "takes": True,
340 "took": True,
341 "taken": True,
342 "need": True,
343 "needing": True,
344 "needs": True,
345 "needed": True,
346 "use": True,
347 "using": True,
348 "uses": True,
349 "used": True,
350 "want": True,
351 "wanna": True,
352 "wanting": True,
353 "wants": True,
```

```

354 "let": True,
355 "lets": True,
356 "letting": True,
357 "let's": True,
358 "suppose": True,
359 "supposing": True,
360 "supposes": True,
361 "supposed": True,
362 "seem": True,
363 "seeming": True,
364 "seems": True,
365 "seemed": True,
366 "say": True,
367 "saying": True,
368 "says": True,
369 "said": True,
370 "know": True,
371 "knowing": True,
372 "knows": True,
373 "knew": True,
374 "known": True,
375 "look": True,
376 "looking": True,
377 "looked": True,
378 "think": True,
379 "thinking": True,
380 "thinks": True,
381 "thought": True,
382 "feel": True,
383 "feels": True,
384 "felt": True,
385 "based": True,
386 "put": True,
387 "puts": True,
388 "begin": True,
389 "began": True,
390 "begun": True,
391 "begins": True,
392 "wanted": True,
393 "like": True,
394 "feel": True,
395 "believe": True,
396 "understand": True,
397 "shall": True,
398 "regard": True,
399 "regards": True,
400 "regarding": True,
401 # Question words and associated words
402 "who": True,
403 "who's": True,
404 "who've": True,
405 "who'd": True,
406 "whoever": True,
407 "whoever's": True,
408 "whom": True,
409 "whomever": True,
410 "whomever's": True,
411 "whose": True,
412 "whosever": True,
413 "whosever's": True,
414 "when": True,
415 "whenever": True,
416 "which": True,
417 "whichever": True,

```

```

418 "where": True,
419 "where's": True,
420 "where'd": True,
421 "wherever": True,
422 "why": True,
423 "why's": True,
424 "why'd": True,
425 "whyever": True,
426 "what": True,
427 "what's": True,
428 "whatever": True,
429 "whence": True,
430 "how": True,
431 "how's": True,
432 "how'd": True,
433 "however": True,
434 "whether": True,
435 "whatsoever": True,
436 # Connector words and irrelevant adverbs
437 "and": True,
438 "or": True,
439 "not": True,
440 "because": True,
441 "also": True,
442 "always": True,
443 "never": True,
444 "only": True,
445 "really": True,
446 "very": True,
447 "greatly": True,
448 "extremely": True,
449 "somewhat": True,
450 "no": True,
451 "nope": True,
452 "nah": True,
453 "yes": True,
454 "yep": True,
455 "yeh": True,
456 "yeah": True,
457 "maybe": True,
458 "perhaps": True,
459 "more": True,
460 "most": True,
461 "less": True,
462 "least": True,
463 "good": True,
464 "great": True,
465 "well": True,
466 "better": True,
467 "best": True,
468 "bad": True,
469 "worse": True,
470 "worst": True,
471 "too": True,
472 "thru": True,
473 "though": True,
474 "although": True,
475 "yet": True,
476 "already": True,
477 "then": True,
478 "even": True,
479 "now": True,
480 "sometimes": True,
481 "still": True,

```

```
482 "together": True,
483 "altogether": True,
484 "entirely": True,
485 "fully": True,
486 "entire": True,
487 "whole": True,
488 "completely": True,
489 "utterly": True,
490 "seemingly": True,
491 "apparently": True,
492 "clearly": True,
493 "obviously": True,
494 "actually": True,
495 "actual": True,
496 "usually": True,
497 "usual": True,
498 "literally": True,
499 "honestly": True,
500 "absolutely": True,
501 "definitely": True,
502 "generally": True,
503 "totally": True,
504 "finally": True,
505 "basically": True,
506 "essentially": True,
507 "fundamentally": True,
508 "automatically": True,
509 "immediately": True,
510 "necessarily": True,
511 "primarily": True,
512 "normally": True,
513 "perfectly": True,
514 "constantly": True,
515 "particularly": True,
516 "eventually": True,
517 "hopefully": True,
518 "mainly": True,
519 "typically": True,
520 "specifically": True,
521 "differently": True,
522 "appropriately": True,
523 "plenty": True,
524 "certainly": True,
525 "unfortunately": True,
526 "ultimately": True,
527 "unlikely": True,
528 "likely": True,
529 "potentially": True,
530 "fortunately": True,
531 "personally": True,
532 "directly": True,
533 "indirectly": True,
534 "nearly": True,
535 "closely": True,
536 "slightly": True,
537 "probably": True,
538 "possibly": True,
539 "especially": True,
540 "frequently": True,
541 "thankfully": True,
542 "often": True,
543 "oftentimes": True,
544 "seldom": True,
545 "rarely": True,
```

```

546 "sure": True,
547 "while": True,
548 "whilst": True,
549 "able": True,
550 "unable": True,
551 "else": True,
552 "ever": True,
553 "once": True,
554 "twice": True,
555 "thrice": True,
556 "almost": True,
557 "again": True,
558 "instead": True,
559 "next": True,
560 "previous": True,
561 "unless": True,
562 "somehow": True,
563 "anyhow": True,
564 "anywhere": True,
565 "somewhere": True,
566 "everywhere": True,
567 "elsewhere": True,
568 "anytime": True,
569 "nowhere": True,
570 "further": True,
571 "anymore": True,
572 "later": True,
573 "ago": True,
574 "ahead": True,
575 "just": True,
576 "same": True,
577 "different": True,
578 "big": True,
579 "small": True,
580 "little": True,
581 "tiny": True,
582 "large": True,
583 "huge": True,
584 "pretty": True,
585 "mostly": True,
586 "anyway": True,
587 "anyways": True,
588 "otherwise": True,
589 "regardless": True,
590 "needless": True,
591 "throughout": True,
592 "additionally": True,
593 "moreover": True,
594 "furthermore": True,
595 "therefore": True,
596 "thereof": True,
597 "meanwhile": True,
598 "likewise": True,
599 "afterwards": True,
600 "nice": True,
601 "nicer": True,
602 "nicest": True,
603 "glad": True,
604 "fine": True,
605 # Irrelevant nouns
606 "thing": True,
607 "thing's": True,
608 "things": True,
609 "stuff": True,

```

```

610 "other's": True,
611 "others": True,
612 "another's": True,
613 "total": True,
614 "true": True,
615 "false": True,
616 "none": True,
617 "way": True,
618 "kind": True,
619 # Lettered numbers and order
620 "zero": True,
621 "zeros": True,
622 "zeroes": True,
623 "one": True,
624 "ones": True,
625 "two": True,
626 "three": True,
627 "four": True,
628 "five": True,
629 "six": True,
630 "seven": True,
631 "eight": True,
632 "nine": True,
633 "ten": True,
634 "twenty": True,
635 "thirty": True,
636 "forty": True,
637 "fifty": True,
638 "sixty": True,
639 "seventy": True,
640 "eighty": True,
641 "ninety": True,
642 "hundred": True,
643 "hundreds": True,
644 "thousand": True,
645 "thousands": True,
646 "million": True,
647 "millions": True,
648 "first": True,
649 "last": True,
650 "second": True,
651 "third": True,
652 "fourth": True,
653 "fifth": True,
654 "sixth": True,
655 "seventh": True,
656 "eighth": True,
657 "ninth": True,
658 "tenth": True,
659 "firstly": True,
660 "secondly": True,
661 "thirdly": True,
662 "lastly": True,
663 # Greetings and slang
664 "hello": True,
665 "hi": True,
666 "hey": True,
667 "sup": True,
668 "yo": True,
669 "greetings": True,
670 "please": True,
671 "okay": True,
672 "ok": True,
673 "y'all": True,

```



```

674 "lol": True,
675 "rofl": True,
676 "thank": True,
677 "thanks": True,
678 "alright": True,
679 "kinda": True,
680 "dont": True,
681 "sorry": True,
682 "idk": True,
683 "doesnt": True,
684 "doesn": True,
685 "didn": True,
686 "didnt": True,
687 "haven": True,
688 "havent": True,
689 "ugh": True,
690 "guess": True,
691 "bullshit": True,
692 "yup": True,
693 "yep": True,
694 "haha": True,
695 "hahaha": True,
696 "hahahaha": True,
697 "hehe": True,
698 "hehehe": True,
699 "till": True,
700 "sure": True,
701 "soon": True,
702 "nah": True,
703 "meh": True,
704 "imo": True,
705 "imho": True,
706 "ill": True,
707 "hella": True,
708 "btw": True,
709 "bro": True,
710
711 # Miscellaneous
712 "www": True,
713 "https": True,
714 "http": True,
715 "com": True,
716 "etc": True,
717 "html": True,
718 "reddit": True,
719 "subreddit": True,
720 "subreddits": True,
721 "comments": True,
722 "reply": True,
723 "replies": True,
724 "thread": True,
725 "threads": True,
726 "post": True,
727 "posts": True,
728 "website": True,
729 "websites": True,
730 "web site": True,
731 "web sites": True
732 }
733
734 """## Download Stop Words"""
735
736 #non custom stop words
737 #from prewritten imported words

```

```

738 import nltk
739 from gensim.parsing.preprocessing import remove_stopwords
740 from nltk.corpus import words
741 from nltk.corpus import stopwords
742 nltk.download('words')
743 nltk.download('stopwords')
744 "using" in words.words()
745 import nltk
746
747 print(stopwords.words('english'))
748 numpy_matrix = df.to_numpy()
749 print(numpy_matrix.shape)
750 print(numpy_matrix[1,1])
751 print("---")
752
753 """## Parse Data
754
755 Processes punctuation, code syntax, upper/lowercase...
756 """
757
758 #filtering out stop words and symbols
759
760
761 apostrophe = 0
762 y = []
763 X = []
764
765 #custom symbols to be removed
766 #of these symbols the only one to keep was <> because that is common mainly in
767 ↳ Javascript so it was kept
768 stop_chars =
769 ↳ ['!',',','.','"', '?', '!', ')', '(', '[', ']', '{', '}', '/', '-', ':', '|', '*', '^', '%', '$', '#', '@', '=', '+', '_',
770 ↳ "\x94", '&']
771 keep_chars = ['<', '>']
772 keep_no_space = ['\x92', '"']
773 apostrophe_remove = 0
774
775 #some simple lemmetization is being done here aswell by removed everything
776 ↳ between apostrophes and spaces
777 #everything is broken down and then reassembled around spaces after symbols
778 ↳ are removed
779 #those words are then passed through the stop words lists
780 for val in range(numpy_matrix.shape[0]):
781     current_phrase = numpy_matrix[val,0]
782     spaced_list = ""
783
784     for i in range(len(current_phrase)):
785         letter = current_phrase[i]
786         if letter.isdigit():
787             pass
788         elif letter in keep_no_space:
789             apostrophe = 1
790             elif apostrophe == 1 and letter == " ":
791                 apostrophe = 0
792             elif apostrophe == 1:
793                 pass
794             elif not letter.isalpha() and letter not in stop_chars and letter not in
795                 ↳ keep_no_space:
796                 spaced_list += " "
797                 spaced_list += letter
798                 spaced_list += " "
799             elif not letter.isalpha() and letter in stop_chars:
800                 spaced_list += " "
801             else:

```

```

796         if letter.isalpha():
797             spaced_list += letter.lower()
798         else:
799             spaced_list += letter
800
801 word_list = []
802
803 stop_words = set(stopwords.words('english'))
804 word_tokens = word_tokenize(spaced_list)
805
806 filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
807
808 filtered_sentence = []
809 my_stop_words = text.ENGLISH_STOP_WORDS
810 filtered = []
811 filtered_string = ""
812 for w in spaced_list.split():
813     if w not in stop_words and w not in my_stop_words
814         ↪ and len(w) > 1:
815         filtered.append(w)
816         filtered_string += " "
817         filtered_string += w
818     #final phrases
819 X.append(filtered_string)
820
821 #exact same being done just for the test set
822 y_test = []
823 x_test = []
824 apostrophe = 0
825 for val in range(test_set.shape[0]):
826     current_phrase = test_set[val]
827     spaced_list = ""
828     #stop_chars =
829     ↪ [' ', ',', '.', '!', '?', '!', ')', '(', '[', ']', '{', '}', '/', '-', ':', '|', '*', '^', '%', '$', '#', '@', '=', '+', '_']
830     ↪ "\x94", '&']
831     #keep_chars = ['<', '>']
832     #keep_no_space = ['\x92', '"']
833 for i in range(len(current_phrase)):
834     letter = current_phrase[i]
835     if letter.isnumeric():
836         pass
837     elif letter in keep_no_space:
838         apostrophe = 1
839     elif apostrophe == 1 and letter == " ":
840         apostrophe = 0
841     elif apostrophe == 1:
842         pass
843     elif not letter.isalpha() and letter not in stop_chars and letter not in
844     ↪ keep_no_space:
845         spaced_list += " "
846         spaced_list += letter
847         spaced_list += " "
848     elif not letter.isalpha() and letter in stop_chars:
849         spaced_list += " "
850     else:
851         if letter.isalpha():
852             spaced_list += letter.lower()
853         else:
854             spaced_list += letter
855
856 word_list = []
857
858 stop_words = set(stopwords.words('english'))
859 word_tokens = word_tokenize(spaced_list)

```

```

856
857     filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
858
859     filtered_sentence = []
860     my_stop_words = text.ENGLISH_STOP_WORDS
861     filtered = []
862     filtered_string = ""
863     for w in spaced_list.split():
864         if w not in stop_words and w not in my_stop_words
865             ↪ and len(w) > 1:
866             filtered.append(w)
867             filtered_string += " "
868             filtered_string += w
869
870     x_test.append(filtered_string)
871     print(X)
872     print(len(X))
873     print(stop_words)
874
875     """## Process muticlass y Vector
876
877     Converts the y text labels to class numbers from 0 - 3. Train test split is
878     ↪ performed at the end
879     """
880
881     import random
882
883     trainingSize = numpy_matrix[:, 0].size
884     y = np.zeros(trainingSize)
885
886     # Convert class labels from values 0 - 3
887     for i in range(trainingSize):
888         if(numpy_matrix[i, 1] == 'Javascript'):
889             y[i] = 0
890         elif(numpy_matrix[i, 1] == 'Matlab'):
891             y[i] = 1
892         elif(numpy_matrix[i, 1] == 'Pytorch'):
893             y[i] = 2
894         elif(numpy_matrix[i, 1] == 'Tensorflow'):
895             y[i] = 3
896
897     # Shuffle data and classes synchronously
898     # Code citation:
899     ↪ https://www.geeksforgeeks.org/python-shuffle-two-lists-with-same-order/
900     tmp = list(zip(X, y))
901     random.shuffle(tmp)
902     tmp_X, tmp_y= zip(*tmp)
903     X, y = list(tmp_X), np.asarray(list(tmp_y))
904
905     """## Evaluation
906
907     Here we plot histograms of each word, as well as the variance of each word's
908     ↪ occurance per class
909     """
910
911     import seaborn as sns
912     import numpy as np
913     import matplotlib.pyplot as plt
914     import pandas as pd
915     import statistics
916
917     Class for evaluating the performance of a logistic regression model.
918     Includes tools for calculating confusion matrix, accuracy, precision,

```

```

916         recall, specificity, and false positive rate.
917
918         Assumes that the input data is the result of a binary logistic regression
919         (e.g. y && y_hat = {0, 1})
920     """
921
922
923     class Evaluation:
924         """
925         Initializes the evaluation from a vector of predicted binary values (y)
926         and a vector of actual values (y_hat). Stores these values in a
927         confusion matrix variable (cm) as well as individual cell
928         values (tp, tn, fp, fn)
929         """
930
931         def __init__(self,
932                     X: np.ndarray,
933                     y: np.ndarray,
934                     word_list: int,
935                     num_classes: int):
936             self.X = X
937             self.y = y
938             self.word_list = word_list
939             self.num_classes = num_classes
940             self.df_trunc = None
941             self.df_trunc_long = None
942
943         """
944         Prints a heatmap of the confusion matrix
945         """
946
947         def confusion_matrix(self, y_hat: np.ndarray):
948             size = y_hat.size
949             cm = np.ndarray(shape = (self.num_classes, self.num_classes))
950             for i in range(size):
951                 x_index = y_hat[i].astype(int)
952                 y_index = y[i].astype(int)
953                 cm[x_index, y_index] += 1
954
955             # Normalize confusion matrix
956             cm = np.divide(cm, np.array(size))
957
958             for i in range(self.num_classes):
959                 for j in range(self.num_classes):
960                     cm[i, j] = round(cm[i, j], 2)
961
962             labels = ["JavaScript", "MATLAB", "PyTorch", "TensorFlow"]
963             df = pd.DataFrame(cm, index = labels, columns = labels)
964             fig = plt.figure(figsize = (10, 10))
965             cell_labels = cm
966             sns.heatmap(df, fmt='', annot=cell_labels)
967             plt.savefig("Heatmap.png")
968             return cm
969
970         """
971         Returns some histograms on word count
972         """
973
974         def tables(self, X, y):
975             # In this data set, the number of disease patients is equal to the
976             # ↪ number of healthy patients
977             kwargs = dict(line_kws = {'lw': 3})
978
979             # Plot all distributions in the data set

```

```

979 X_tmp_JavaScript = []
980 X_tmp_MATLAB = []
981 X_tmp_PyTorch = []
982 X_tmp_TF = []
983 X_ord = np.sum(X, axis = 0)
984
985 for i in range(X[:, 0].size):
986     if(y[i] == 0):
987         X_tmp_JavaScript.append(X[i, :])
988     elif(y[i] == 1):
989         X_tmp_MATLAB.append(X[i, :])
990     elif(y[i] == 2):
991         X_tmp_PyTorch.append(X[i, :])
992     elif(y[i] == 3):
993         X_tmp_TF.append(X[i, :])
994
995 X_tmp_JavaScript = np.sum(X_tmp_JavaScript, axis = 0)
996 X_tmp_MATLAB = np.sum(X_tmp_MATLAB, axis = 0)
997 X_tmp_PyTorch = np.sum(X_tmp_PyTorch, axis = 0)
998 X_tmp_TF = np.sum(X_tmp_TF, axis = 0)
999
1000 variance = []
1001 for i in range(len(self.word_list)):
1002     values = [X_tmp_JavaScript[i], X_tmp_MATLAB[i], X_tmp_PyTorch[i],
1003             ↪ X_tmp_TF[i]]
1004     variance.append(statistics.variance(values))
1005
1006 table = {'Word': self.word_list,
1007         'JavaScript Frequency': X_tmp_JavaScript,
1008         'MATLAB Frequency': X_tmp_MATLAB,
1009         'PyTorch Frequency': X_tmp_PyTorch,
1010         'TensorFlow Frequency': X_tmp_TF,
1011         'Total Frequency': X_ord,
1012         'Variance': variance}
1013
1014 df = pd.DataFrame(data = table)
1015
1016 self.df_trunc = df.iloc[0:20]
1017
1018 self.df_trunc_long = df.iloc[0:100]
1019
1020 return df
1021
1022 def histograms(self):
1023     fig = plt.figure(figsize = (20, 5))
1024     plt.savefig("JS.png")
1025
1026     sns.barplot(data = self.df_trunc,
1027                 x = "Word",
1028                 y = "JavaScript Frequency",
1029                 palette = "CMRmap_r")
1030
1031     fig = plt.figure(figsize = (20, 5))
1032     plt.savefig("MATLAB.png")
1033
1034     sns.barplot(data = self.df_trunc,
1035                 x = "Word",
1036                 y = "MATLAB Frequency",
1037                 palette = "CMRmap_r")
1038
1039     fig = plt.figure(figsize = (20, 5))
1040     plt.savefig("PyTorch.png")
1041

```

```

1042     sns.barplot(data = self.df_trunc,
1043                 x = "Word",
1044                 y = "PyTorch Frequency",
1045                 palette = "CMRmap_r")
1046
1047     fig = plt.figure(figsize = (20, 5))
1048     plt.savefig("TF.png")
1049
1050     sns.barplot(data = self.df_trunc,
1051                 x = "Word",
1052                 y = "TensorFlow Frequency",
1053                 palette = "CMRmap_r")
1054
1055     self.df_trunc_long = self.df_trunc_long.sort_values(by = 'Variance',
1056                                                         ↪ ascending = False)
1057
1058     sns.barplot(data = self.df_trunc_long,
1059                 x = "Word",
1060                 y = "Variance")
1061
1062     """# Naive Bayes Implementation"""
1063
1064     !pip install tqdm
1065
1066     import math
1067     from tqdm import tqdm
1068     from sklearn.model_selection import train_test_split
1069     from sklearn.feature_extraction.text import TfidfTransformer
1070     from sklearn.preprocessing import Normalizer
1071     import nltk
1072     from nltk.stem import WordNetLemmatizer
1073     from nltk.corpus import stopwords
1074
1075     nltk.download('omw-1.4')
1076
1077     """
1078     Implements a MultiClass Naive Bayes Classifier
1079     """
1080     class NaiveBayes:
1081
1082         """
1083         Initializes the model with
1084
1085         num_classes - Number of classes used in the output data
1086         stop_words - A set containing stop words to be filtered out by the
1087         ↪ vectorizer
1088         """
1089         def __init__(self, num_classes, stop_words):
1090             self.num_classes = num_classes
1091             self.theta = np.zeros(self.num_classes)
1092             self.word_list= []
1093             self.stop_words = stop_words
1094             self.num_words= 0
1095             self.theta_j0 = None
1096             self.theta_j1 = None
1097             self.eval = None
1098             self.vocab = None
1099
1100         """
1101         Static implemmtation of sigmoid used for predicting final output
1102         """
1103         @staticmethod
1104         def __sigmoid(x):
1105             return 1 / (1 + math.exp(-x))

```

```

1104
1105 @staticmethod
1106 def __getvocab(self, X, y):
1107     # First vectorize the input dataset to obtain a histogram of words
1108     vectorizer = CountVectorizer(max_features = 1200,
1109                                binary = False,
1110                                ngram_range = (1,3),
1111                                stop_words = self.stop_words)
1112
1113     tfidf_transformer = TfidfTransformer(smooth_idf = True,
1114                                         use_idf = True)
1115
1116     normalizer = Normalizer()
1117
1118     for i in range(y.size):
1119         sentences = nltk.sent_tokenize(X[i])
1120         lemmatizer = WordNetLemmatizer()
1121
1122         # Lemmatization
1123         for j in range(len(sentences)):
1124             words = nltk.word_tokenize(sentences[j])
1125             words = [lemmatizer.lemmatize(word) for word in words if word
1126                     ↪ not in stop_words]
1127
1128             sentences = ' '.join(words)
1129
1130             X[i] = sentences
1131
1132     vectorized_matrix = vectorizer.fit_transform(X)
1133     vectorized_idf = tfidf_transformer.fit_transform(vectorized_matrix)
1134     normalized_idf = normalizer.fit_transform(vectorized_idf)
1135     X = normalized_idf.toarray()
1136
1137     # Obtain the word list, number of words, and empty theta probabilities
1138     ↪ from
1139     # the vectorization
1140     self.word_list = list(vectorizer.get_feature_names_out())
1141     self.num_words = len(self.word_list)
1142     self.theta_j0 = np.zeros((self.num_words, self.num_classes))
1143     self.theta_j1 = np.zeros((self.num_words, self.num_classes))
1144
1145     # Create Evaluation class for new dictionary based on variance
1146     e = Evaluation(X, y, self.word_list, 4)
1147
1148     bayes_data_frame = e.tables(X, y)
1149     bayes_data_frame = bayes_data_frame.sort_values(by = 'Variance',
1150     ↪ ascending = False)
1151
1152     variance_vocab = bayes_data_frame['Word'].iloc[0:130].astype(str)
1153     variance_vocab = variance_vocab.to_list()
1154     vocab = {k: v for v, k in enumerate(variance_vocab)}
1155     return vocab
1156
1157 """
1158 Trains the Naive Bayes classifier using an input of strings and
1159 ↪ corresponding
1160 numeric class labels. The fit method contains an internal instance of
1161 ↪ sklearn's
1162 CountVectorizer, which stores the list of words based on the input data
1163 ↪ and
1164 the stop words.
1165
1166 X: A list of strings containing each block of text in the dataset
1167 y: A numpy array of numeric class labels corresponding to the items in X

```



```

1162     """
1163     def fit(self, X, y, plot_flag):
1164
1165         vocab = self.__getvocab(self, X, y)
1166
1167         print(vocab)
1168
1169         # First vectorize the input dataset to obtain a histogram of words
1170         vectorizer = CountVectorizer(max_features = 800,
1171                                     binary = False,
1172                                     ngram_range = (1,3),
1173                                     vocabulary = vocab,
1174                                     stop_words = self.stop_words)
1175
1176         tf_idf_transformer = TfidfTransformer(smooth_idf = True,
1177                                               use_idf = True)
1178
1179         normalizer = Normalizer()
1180
1181         for i in range(y.size):
1182             sentences = nltk.sent_tokenize(X[i])
1183             lemmatizer = WordNetLemmatizer()
1184
1185             # Lemmatization
1186             for j in range(len(sentences)):
1187                 words = nltk.word_tokenize(sentences[j])
1188                 words = [lemmatizer.lemmatize(word) for word in words if word
1189                          ↪ not in stop_words]
1189
1190             sentences = ' '.join(words)
1191
1192             X[i] = sentences
1193
1194         vectorized_matrix = vectorizer.fit_transform(X)
1195         vectorized_idf = tf_idf_transformer.fit_transform(vectorized_matrix)
1196
1197         # Obtain the word list, number of words, and empty theta probabilities
1198         ↪ from
1199         # the vectorization
1200         self.word_list = list(vectorizer.get_feature_names_out())
1201         self.num_words = len(self.word_list)
1202
1203         normalized_idf = normalizer.fit_transform(vectorized_idf)
1204         X = normalized_idf.toarray()
1205
1206         self.theta_j0 = np.zeros((self.num_words, self.num_classes))
1207         self.theta_j1 = np.zeros((self.num_words, self.num_classes))
1208
1209         # Transform y into an n-dimensional array where n = num_classes, and
1210         # convert each dimension values into binary class values (one class
1211         ↪ against all others)
1212         y_multiclass = np.zeros((self.num_classes, y.size))
1213         for n in range(self.num_classes):
1214             for i in range(y.size):
1215                 y_multiclass[n, i] = 1 if (y[i] == n) else 0
1216         specials = ["tf", "tensorflow", "pytorch", "torch"]
1217         weightfactor = 100
1218         # Train each class' binary theta feature values
1219         for n in range(self.num_classes):
1220             self.theta[n] = y_multiclass[n, :].sum() / y.size
1221             for i in tqdm(range(self.num_words),
1222                           desc="Class: " + str(n + 1) + " / " +
1223                           ↪ str(self.num_classes),

```

```

1222         ascii=False,
1223         ncols=75):
1224     for j in range(y.size):
1225         if(y_multiclass[n, j] == 1):
1226             self.theta_j1[i, n] += X[j, i]
1227         else:
1228             self.theta_j0[i, n] += X[j, i]
1229
1230     # Dividing by total number of items of each class + Laplace
1231     ↪ Smoothing
1232     self.theta_j1[i, n] = (self.theta_j1[i, n] + 1) / (y_multiclass[n,
1233     ↪ :].sum() + 2)
1234     self.theta_j0[i, n] = (self.theta_j0[i, n] + 1) / ((1 -
1235     ↪ y_multiclass[n, :]).sum() + 2)
1236
1237     if(plot_flag):
1238         eval = Evaluation(X, y, self.word_list, 4)
1239         eval.tables(X, y)
1240         eval.histograms()
1241
1242     """
1243     Classify a new input x as one of the n classes. Predicted class is based
1244     on the values derived from the fit function above
1245
1246     x - String containing an unlabeled block of text
1247     """
1248     def predict(self, x):
1249         bias = np.zeros(self.num_classes)
1250         xTw = np.zeros(self.num_classes)
1251         result = np.zeros(self.num_classes)
1252
1253         # Calculate weights and bias for each class
1254         for n in range(self.num_classes):
1255             bias[n] = math.log(self.theta[n] / (1 - self.theta[n]))
1256             hist = np.vstack((self.theta_j0[:, n], self.theta_j1[:, n]))
1257
1258         # Check each word in the class' word list and see if it
1259         # exists in the input block of text
1260         for i in range(self.num_words):
1261             currentWord = self.word_list[i]
1262             if(x.count(currentWord)):
1263                 xTw[n] += math.log(hist[1, i] / hist[0, i])
1264             else:
1265                 xTw[n] += math.log((1 - hist[1, i]) / (1 - hist[0, i]))
1266
1267         # Pass the resulting bias and weights into the sigmoid function
1268         result[n] = self._sigmoid(bias[n] + xTw[n])
1269
1270         # Choose the max result
1271         result_max = np.where(result == max(result))[0][0]
1272
1273         return result_max
1274
1275     import math
1276     from tqdm import tqdm
1277     from sklearn.model_selection import train_test_split
1278     from sklearn.feature_extraction.text import TfidfTransformer
1279     from sklearn.preprocessing import Normalizer
1280     import nltk
1281     from nltk.stem import WordNetLemmatizer
1282     from nltk.corpus import stopwords
1283
1284     nltk.download('omw-1.4')

```

```

1283 """
1284 Implements a MultiClass vectorizer
1285 This is effectively the same code as the naive bayes minus the actual naive
1286 bayes
1287 Just the data processing is included for the vectorization so that
1288 """
1289 class simple_vectorizer():
1290
1291     """
1292     Initializes the model with
1293
1294     num_classes - Number of classes used in the output data
1295     stop_words - A set containing stop words to be filtered out by the
1296     vectorizer
1297     """
1298     def __init__(self, num_classes, stop_words):
1299         self.num_classes = num_classes
1300         self.theta = np.zeros(self.num_classes)
1301         self.word_list= []
1302         self.stop_words = stop_words
1303         self.num_words= 0
1304         self.theta_j0 = None
1305         self.theta_j1 = None
1306         self.eval = None
1307         self.vocab = None
1308
1309     """
1310     Static implemmtation of sigmoid used for predicting final output
1311     """
1312     @staticmethod
1313     def __sigmoid(x):
1314         return 1 / (1 + math.exp(-x))
1315
1316     """
1317     Trains the Naive Bayes classifier using an input of strings and
1318     corresponding
1319     numeric class labels. The fit method contains an internal instance of
1320     sklearn's
1321     CountVectorizer, which stores the list of words based on the input data
1322     and
1323     the stop words.
1324
1325     X: A list of strings containing each block of text in the dataset
1326     y: A numpy array of numeric class labels corresponding to the items in X
1327     """
1328     def train_vector(self, X, y):
1329         # First vectorize the input dataset to obtain a histogram of words
1330         vectorizer = CountVectorizer(max_features = 800,
1331                                     binary = False,
1332                                     ngram_range = (1,3),
1333                                     stop_words = self.stop_words)
1334
1335         tf_idf_transformer = TfidfTransformer(smooth_idf = True,
1336                                                use_idf = True)
1337
1338         normalizer = Normalizer()
1339
1340         for i in range(y.size):
1341             sentences = nltk.sent_tokenize(X[i])
1342             lemmatizer = WordNetLemmatizer()
1343
1344             # Lemmatization

```

```

1342         for j in range(len(sentences)):
1343             words = nltk.word_tokenize(sentences[j])
1344             words = [lemmatizer.lemmatize(word) for word in words if word
1345                       ↪ not in stop_words]
1346
1347         sentences = ' '.join(words)
1348
1349         X[i] = sentences
1350
1351     vectorized_matrix = vectorizer.fit_transform(X)
1352     vectorized_idf = tf_idf_transformer.fit_transform(vectorized_matrix)
1353     normalized_idf = normalizer.fit_transform(vectorized_idf)
1354     X = normalized_idf.toarray()
1355
1356     # Obtain the word list, number of words, and empty theta probabilities
1357     ↪ from
1358     # the vectorization
1359     self.word_list = list(vectorizer.get_feature_names_out())
1360     self.num_words = len(self.word_list)
1361     return X
1362
1363 def test_vector(self, X):
1364     # First vectorize the input dataset to obtain a histogram of words
1365     vectorizer = CountVectorizer(max_features = 800,
1366                                 binary = False,
1367                                 ngram_range = (1,3),
1368                                 vocabulary = self.word_list,
1369                                 stop_words = self.stop_words)
1370
1371     tf_idf_transformer = TfidfTransformer(smooth_idf = True,
1372                                           use_idf = True)
1373
1374     normalizer = Normalizer()
1375
1376     for i in range(len(X)):
1377         sentences = nltk.sent_tokenize(X[i])
1378         lemmatizer = WordNetLemmatizer()
1379
1380         # Lemmatization
1381         for j in range(len(sentences)):
1382             words = nltk.word_tokenize(sentences[j])
1383             words = [lemmatizer.lemmatize(word) for word in words if word
1384                       ↪ not in stop_words]
1385
1386         sentences = ' '.join(words)
1387
1388         X[i] = sentences
1389
1390         vectorized_matrix = vectorizer.fit_transform(X)
1391         vectorized_idf = tf_idf_transformer.fit_transform(vectorized_matrix)
1392         normalized_idf = normalizer.fit_transform(vectorized_idf)
1393         X = normalized_idf.toarray()
1394         return X
1395
1396 """#Train A K-Fold Classifier"""
1397
1398 import pandas as pd, numpy as np
1399 from sklearn.preprocessing import StandardScaler
1400 """
1401     Class for performing K-fold cross validation and
1402     returning its mean error. Ideally used for comparing
1403     the performance of multiple logistic regression models.
1404 """
1405 class KFold:

```

```

1403     """
1404     Initializes class by shuffling the input data and obtaining
1405     a validation set size based on the input dimensions and specified
1406     K value.
1407
1408     """
1409     def __init__(self,
1410                 X: np.ndarray,
1411                 y: np.ndarray,
1412                 k: int,
1413                 stop_words: set):
1414         # Initialized data and classes
1415         self.X = X
1416         self.y = y
1417
1418         self.k = k
1419
1420         self.stop_words = stop_words
1421
1422         # Validation size set = number of rows / k
1423         self.validation_set_size = int(len(self.X) / k)
1424
1425     @staticmethod
1426     def __evaluate(model, X_val, Y_val):
1427         accuracy = 0
1428         total = 0
1429         #print(X_val)
1430         for i in tqdm(range(len(X_val)),
1431                       desc="Predicting Test Data: ",
1432                       ascii=False,
1433                       ncols=75):
1434
1435             prediction = model.predict(X_val[i])
1436             if prediction == Y_val[i]:
1437                 accuracy += 1
1438             total += 1
1439
1440         accuracy_percent = (accuracy / total) * 100
1441         accuracy_percent = float(f'{accuracy_percent:.2f}')
1442
1443         print("\n\n ===== \n\n")
1444         print("Accuracy: ", accuracy_percent, "%")
1445
1446         return (accuracy / total)
1447
1448     """
1449     Performs the cross validation on K iterations of the input data.
1450     The cross validation is performed by taking the first validation
1451     set from the top of the input data and then subsequently shifting
1452     (rolling) the input data N elements, where N = validation set size.
1453
1454     Note that this method of cycling and partitioning will automatically
1455     throw any remainder of input data into the validation set if the data
1456     set cannot be evenly divided into K sets.
1457
1458     """
1459     def cross_validation(self):
1460         # Initialize error count
1461         self.test_acc = 0
1462
1463         for i in range(self.k):
1464             print("Fold - ", i + 1, " / ", self.k)
1465

```

```

1466         # Training set = all rows where index is larger than validation
1467         ↪ set size
1468         X_train_fold = self.X[self.validation_set_size:]
1469         Y_train_fold = self.y[self.validation_set_size:]
1470
1471         # Test set = all rows where index is below validation set size
1472         X_validation_fold = self.X[:self.validation_set_size]
1473         Y_validation_fold = self.y[:self.validation_set_size]
1474
1475         # Implement the Naive Bayes model
1476         bayes = NaiveBayes(4, stop_words)
1477
1478         bayes.fit(X_train_fold, Y_train_fold, 0)
1479
1480         # Evaluate the predicted Y with the actual Y from the test data
1481         self.test_eval = self.__evaluate(bayes, X_validation_fold,
1482         ↪ Y_validation_fold)
1483
1484         # Accumulate the error
1485         self.test_acc += self.test_eval
1486
1487         # Shift the X data over a validation set size to ensure a new
1488         # validation set data for the next training iteration
1489         self.X = np.roll(self.X, -self.validation_set_size, axis = 0)
1490         self.y = np.roll(self.y, -self.validation_set_size, axis = 0)
1491
1492         # Normalize error of all iterations of self, the validation set
1493         self.test_acc /= self.k
1494
1495         print("-----")
1496
1497         total_acc = self.test_acc * 100
1498
1499         print("Total Accuracy: ",
1500               float(f'{total_acc:.2f}'), "%")
1501
1502         return(total_acc)
1503
1504 import time
1505 class KFold_bag:
1506     """
1507     Initializes class by shuffling the input data and obtaining
1508     a validation set size based on the input dimensions and specified
1509     K value.
1510
1511     """
1512     def __init__(self,
1513                 X: np.ndarray,
1514                 y: np.ndarray,
1515                 k: int,
1516                 stop_words: set):
1517         # Initialized data and classes
1518         self.X = X
1519         self.y = y
1520
1521         self.k = k
1522
1523         self.stop_words = stop_words
1524         self.bags = []
1525         self.bag_size = 10
1526
1527         # Validation size set = number of rows / k
1528         self.validation_set_size = int(len(self.X) / k)

```

```

1528
1529 @staticmethod
1530 def __evaluate(model, X_val, Y_val):
1531     accuracy = 0
1532     total = 0
1533     #print(X_val)
1534     for i in tqdm(range(len(X_val)),
1535                   desc="Predicting Test Data: ",
1536                   ascii=False,
1537                   ncols=75):
1538         blanks = []
1539
1540         for j in range(10):
1541             val = model[j].predict(X_val[i])
1542             blanks.append(val)
1543             print("blanks", blanks, Y_val[i])
1544             top = 0
1545             top_instance = 0
1546             for j in range(10):
1547                 if top_instance > blanks.count(j):
1548                     pass
1549                 else:
1550                     top_instance = blanks.count(j)
1551                     top = j
1552
1553             if top == Y_val[i]:
1554                 accuracy += 1
1555
1556             total += 1
1557
1558     accuracy_percent = (accuracy / total) * 100
1559     accuracy_percent = float(f'{accuracy_percent:.2f}')
1560
1561     print("\n\n ===== \n\n")
1562     print("Accuracy: ", accuracy_percent, "%")
1563
1564     return (accuracy / total)
1565
1566 """
1567     Performs the cross validation on K iterations of the input data.
1568     The cross validation is performed by taking the first validation
1569     set from the top of the input data and then subsequently shifting
1570     (rolling) the input data N elements, where N = validation set size.
1571
1572     Note that this method of cycling and partitioning will automatically
1573     throw any remainder of input data into the validation set if the data
1574     set cannot be evenly divided into K sets.
1575 """
1576 def cross_validation(self):
1577     # Initialize error count
1578     self.test_acc = 0
1579
1580     for i in range(self.k):
1581         print("Fold - ", i + 1, " / ", self.k)
1582
1583         # Training set = all rows where index is larger than validation
1584         # ↪ set size
1585         X_train_fold = self.X[self.validation_set_size:]
1586         Y_train_fold = self.y[self.validation_set_size:]
1587
1588         # Test set = all rows where index is below validation set size
1589         X_validation_fold = self.X[:self.validation_set_size]
1590         Y_validation_fold = self.y[:self.validation_set_size]
1591         #print(X_validation_fold)

```

```

1591         for j in range(self.bag_size):
1592
1593             #percentage = random.uniform(0.75, 0.97)
1594             percentage = 0.99
1595             percentage = int(round(percent*100))/100
1596             #print(i,percentage)
1597             bag_X_train, bag_X_test, bag_y_train, bag_y_test =
1598                 ↪ train_test_split(X_validation_fold, Y_validation_fold,
1599                 ↪ test_size=1-percentage, random_state=30)
1600             #print(bag_X_train)
1601             #print(bag_X_train.shape)
1602             #print(bag_y_train.shape)
1603             bag_bayes = NaiveBayes(4, stop_words)
1604             bag_bayes.fit(bag_X_train, bag_y_train, 0)
1605             self.bags.append(bag_bayes)
1606             #time.sleep(10)
1607             #print(self.bags)
1608             # Evaluate the predicted Y with the actual Y from the test data
1609             self.test_eval = self.__evaluate(self.bags, X_validation_fold,
1610             ↪ Y_validation_fold)
1611
1612             # Accumulate the error
1613             self.test_acc += self.test_eval
1614
1615             # Shift the X data over a validation set size to ensure a new
1616             # validation set data for the next training iteration
1617             self.X = np.roll(self.X, -self.validation_set_size, axis = 0)
1618             self.y = np.roll(self.y, -self.validation_set_size, axis = 0)
1619
1620             # Normalize error of all iterations of self, the validation set
1621             self.test_acc /= self.k
1622
1623             print("-----")
1624
1625             total_acc = self.test_acc * 100
1626
1627             print("Total Accuracy: ",
1628                   float(f'{total_acc:.2f}'), "%")
1629
1630             return(total_acc)
1631
1632     """# Test A K-Fold Classifier
1633
1634     This section of the code just takes the kfold with naive bayes to test
1635     ↪ different presets to determine accuracy.
1636     """
1637
1638     import nltk
1639     nltk.download('wordnet')
1640     kf = KFold(X, y, 10, stop_words)
1641     kf.cross_validation()
1642
1643     print(X)
1644
1645     bayes = NaiveBayes(4, stop_words)
1646     bayes.fit(X, y, 1)
1647
1648     y_pred = []
1649     for i in x_test:
1650         y_pred.append(bayes.predict(i))
1651
1652     output = []
1653
1654     step = 0

```



```

1651 for i in y_pred:
1652     if i == 0:
1653         output.append([step+1, "Javascript"])
1654     elif i == 1:
1655         output.append([step+1, "Matlab"])
1656     elif i == 2:
1657         output.append([step+1, "Pytorch"])
1658     elif i == 3:
1659         output.append([step+1, "Tensorflow"])
1660     step += 1
1661 print(output)
1662 df = pd.DataFrame(output)
1663 df.to_csv('output.csv', index=False)
1664
1665 print(x_test)
1666
1667 """#Other ML test
1668
1669 Vectorization for train and test sets
1670 """
1671
1672 bayes = simple_vectorizer(4, stop_words)
1673 train_vector = bayes.train_vector(X, y)
1674 test_vector = bayes.test_vector(x_test)
1675
1676 print(train_vector)
1677 print(len(train_vector))
1678 print(len(train_vector[0]))
1679
1680 print(test_vector)
1681
1682 """Random Forest Test"""
1683
1684 from sklearn.ensemble import RandomForestClassifier
1685 from sklearn.datasets import make_classification
1686 from sklearn.metrics import accuracy_score
1687 rf_X_train, rf_X_test, rf_y_train, rf_y_test = train_test_split(train_vector,
1688     ↪ y, test_size=0.2)
1689 clf = RandomForestClassifier(max_depth=20, random_state=0)
1689 clf.fit(rf_X_train, rf_y_train)
1690 y_pred = clf.predict(rf_X_test)
1691 print(y_pred)
1692 print(rf_y_test)
1693 print(accuracy_score(rf_y_test, y_pred))
1694
1695 """Support Vector Machine"""
1696
1697 from sklearn import svm
1698 rf_X_train, rf_X_test, rf_y_train, rf_y_test = train_test_split(train_vector,
1699     ↪ y, test_size=0.2)
1700 clf = svm.SVC()
1700 clf.fit(rf_X_train, rf_y_train)
1701 y_pred = clf.predict(rf_X_test)
1702 print(y_pred)
1703 print(rf_y_test)
1704 print(accuracy_score(rf_y_test, y_pred))
1705
1706 """Adaboost"""
1707
1708 from sklearn.ensemble import AdaBoostClassifier
1709 rf_X_train, rf_X_test, rf_y_train, rf_y_test = train_test_split(train_vector,
1710     ↪ y, test_size=0.2)
1710 clf = AdaBoostClassifier(n_estimators=200, random_state=0)
1711 clf.fit(rf_X_train, rf_y_train)

```

```

1712 y_pred = clf.predict(rf_X_test)
1713 print(y_pred)
1714 print(rf_y_test)
1715 print(accuracy_score(rf_y_test, y_pred))
1716
1717 """Gaussian Naive Bayes"""
1718
1719 from sklearn.naive_bayes import GaussianNB
1720 from sklearn.metrics import accuracy_score
1721 from sklearn.metrics import confusion_matrix
1722 rf_X_train, rf_X_test, rf_y_train, rf_y_test = train_test_split(train_vector,
↪ y, test_size=0.25)
1723 clf = GaussianNB()
1724 clf.fit(rf_X_train, rf_y_train)
1725 y_pred = clf.predict(rf_X_test)
1726 print(y_pred)
1727 print(rf_y_test)
1728 print(accuracy_score(rf_y_test, y_pred))
1729 print(confusion_matrix(rf_y_test, y_pred))
1730
1731 """#Stacking
1732
1733 tested with:
1734 Random Forest,
1735
1736 Support Vector Machine,
1737
1738 Adaboost
1739 """
1740
1741 rf_X_train, rf_X_test, rf_y_train, rf_y_test = train_test_split(train_vector,
↪ y, test_size=0.2, random_state=42)
1742
1743 stack = []
1744
1745 rf = RandomForestClassifier(max_depth=20, random_state=0)
1746 rf.fit(rf_X_train, rf_y_train)
1747 rf_y_pred = rf.predict(rf_X_test)
1748 stack.append(rf_y_pred)
1749
1750 ada = AdaBoostClassifier(n_estimators=200, random_state=0)
1751 ada.fit(rf_X_train, rf_y_train)
1752 y_pred = ada.predict(rf_X_test)
1753 #stack.append(y_pred)
1754
1755 sv = svm.SVC()
1756 sv.fit(rf_X_train, rf_y_train)
1757 y_pred = sv.predict(rf_X_test)
1758 #stack.append(y_pred)
1759
1760 pred = []
1761 for i in range(len(y_pred)):
1762     top = 0
1763     current = 0
1764     current_list = []
1765     for j in range(len(stack)):
1766         current_list.append(stack[j][i])
1767     for j in range(len(stack)):
1768         if top < current_list.count(j):
1769             top = current_list.count(j)
1770             current = j
1771
1772     pred.append(current)
1773

```

```
1774 print(pred)
1775
1776 print(accuracy_score(rf_y_test, pred))
1777
1778 print(accuracy_score(rf_y_test, rf_y_pred))
1779
```