COMP 4900A Mini-Project 2

Connor Stewart (101041125)
Michael Kazman (101037568)
Trevor Johns(101036054)

Majid Komeilli

November 8th, 2020

1 Abstract

The task of this project is to design and implement the well-known Bernoulli Naïve Bayes classifier from scratch, as well as additional classifiers and cross-validation through whatever means necessary. Further analysis and comparison of the chosen classifiers will be discussed concerning accuracy and runtime. The additional algorithms are taken from Python's well-established scikit-learn library. These classifiers are intended to run through a set of training data that consists of Reddit posts along with their associated subreddit. Furthermore, the training set is used through cross-validation to determine classifiers with the highest accuracy. The selected models will then generate predictions on a supplied testing set, which will be posted on Kaggle to determine the most successful groups. It should be noted that the codebase was designed with mathematical and statistical precision in mind. This is essential for replicating the confidence rates associated with the chosen classification algorithms. The program's codebase was built with modularity in mind to allow alternative datasets an easy implementation without risk of overfitting. Following all the aforementioned computations, analysis of the datasets and results will be inferred. In the report herein, we will discuss in rigorous detail the data sets, results, as well as their significance.

2 Introduction

The purpose of this project is to classify a dataset consisting of Reddit posts. This is done using a multitude of classification algorithms, including a Naïve Bayes implementation made from scratch, as well as a select few models from Python's scikit-learn library. The model found to be most accurate will then provide predictions for an unlabelled testing set and be subsequently posted to Kaggle. These predictions will be evaluated on 30% of the testing set, alongside other research groups, to provide a leaderboard score. From this, many important findings have been discovered, such as the importance of data shuffling, ensemble stacking, as well as insights into the specific machine learning methods used. Additionally, further insights into the maximum possible accuracy scores have been discovered and will be discussed in the Discussion and Conclusion section. The analysis of aspects - such as accuracy and runtime - is further discussed in the following paper. To achieve the training and testing necessary for the classifiers, two datasets were used. The training set consists of two columns, the first being a string representation of the Reddit comment or post, and the second being the source (subreddit) it was extracted from. The testing set also consists of two columns. However, the first column is an identification number representing the entry and the second column is the string of text, exactly as seen in the training set's second column. The goal of this application is to choose - from a list of eight unique subreddits - which community the post originated from. The section below speaks of the provided training and testing sets, as well as the pre-processing steps needed for increased scores.

3 Datasets

The training dataset consists of two primary columns, the first being the body of text posted on Reddit, and the second consisting of the subreddit it was extracted from. The entry comes from a list of eight possible subreddits, including rpg (role-playing games), anime (Japanese animation), datascience (matters related to data science), hardware (computer hardware news and discussions), cars (car-related content), gamernews (emerging video game news), gamedev (video game development), and computers (computer-related content). The machine learning model's main purpose is to correctly classify the observations contained within this training set into one of the above subreddit categories.

The testing dataset consists of two primary columns, the first being the identification number of the Reddit entry, and the second being the text contained in that entry. The identification column ("id") consists of simple integer values – which are zero-indexed and arranged in increasing order. The second column of text ("body") consists of text strings randomly taken from one of the listed subreddits. The string contains the body of the uploaded comment or post and can consist of letters, numbers, symbols, and characters with no limitation on length. In essence, the body consists of anything that can be posted on the corresponding subreddit. The dataset contains exactly 2898 lines of text extracted from the subreddits listed in the paragraph. The actual length of the text contained within the "body" column is highly variable, with some entries containing little more than several words and others containing an entire paragraph of text. Data was frequently tested using a reduced excel file consisting of 30 text entries. This was to ensure shorter run times during bug-fixing; however, the final results were generated using the entire dataset.

The training sets were in files named "train.csv" and "train_reduced.csv" for the full and partial training sets, respectively. Likewise, the testing sets were in files named "test.csv" and "test_reduced.csv" for the full and partial training sets, respectively. The training dataset is used for training the models and determining accuracy scores, while the test dataset is used to generate predictions for each post, alongside Kaggle for correctness.

We used a variety of methods on the dataset to replicate the original Bernoulli Naïve Bayes classifier. This involves the use of scikit-learn's CountVectorizer with a binary parameter to determine word occurrence. The binarized CountVectorizer constructs a matrix of 1's and 0's, representing whether a word is present in that row (document). This is used throughout the algorithm to calculate the theta value for each feature in a document. Before training steps, the entire dataset is iterated over to determine the proportion of each class within the dataset. The CountVectorizer natively contains stop word removal as well as the removal of punctuation, Unicode characters, and other unimportant symbols. Typical English stop words such as "the", "and", "a", etc. were removed from the document corpus. This is to allow the algorithm a matrix containing only important features. The information gain with words like "and" and "the" is virtually meaningless, and therefore needed to be removed in the pre-processing stage.

A further pre-processing that could be taken is stemming and lemmatization. These concepts allow more information to be derived from words with common prefixes and suffixes. For example, suffixes like "swim" and "swimming" or "plays" and "played" are all similar in meaning but have slight modifications causing the model to register them as completely separate entries. Allowing the model to recognize these similarities helps improve accuracy. Specifically, stemming is the process by which words are reduced to their root sequence of characters [2]. This would allow words like "burn", "burner", and "burning" to all be recognized as the word "burn". On the other hand, lemmatization is the process of converting a word to its base form rather than simply removing affixes to expose the root of the word [2]. This allows words to be analyzed in finer granularity, thus preventing mistakes stemming commits - such as converting the word "caring" to "car" by removing the -ing suffix directly. Lemmatization does this by taking the context of English words into account. This fixes errors commonly seen when using stemming, like converting the word "hating" to "hat" rather than "hate". We attempted to use external libraries like CoreNLP and spaCy to do this; however, the runtime was drastically increased. Furthermore, the feature list was limited to the 5000 most common words, as words beyond that are often only found once. This is important to consider as words beyond that provide no information gain and yet increase runtime overhead.

In the scikit-learn model, a different approach was taken as other classification algorithms are not impacted as heavily as the Bernoulli Naïve Bayes classifier, concerning unnecessary information. Rather than a binarized vectorizer, included in this model is the use of a Tf-idf vectorizer. This is due to the increase in accuracy provided by the tf-idf weights. Rather than using a simple binary representation, whether occurrence occurred, tf-idf weights provide the matrix representation with more meaning. Namely, a tf-idf weighting scheme evaluates how important a word is in a document concerning the entire corpus. Moreover, shuffling was performed on the dataset to generate random permutations of the dataset and allow for a greater accuracy score. A shuffled dataset helps generate more accurate predictions as the training set is sorted by subreddit. This means that a non-shuffled dataset will contain sections non-representative of the class distribution. Specifically, shuffling helps reduce variance and makes sure the model remains generalized and less prone to overfitting. Alongside stacking, the machine learning model was able to prune out many incorrect predictions and accurately classify the dataset, and the approach of which will be discussed in the following section.

4 Proposed Approach

The application initially made use of the Bernoulli Naïve Bayes classifiers. This classifier is a variant of the Naïve Bayes classification algorithm for Machine Learning. The Naïve Bayes system is based on the Bayes theorem for the likelihood of occurrence of an event [5]. Naïve Bayes is used as a probabilistic classifier to determine the highest likelihood of which class a dataset belongs to [5]. Under the Naïve Bayes classifier, it is important to consider that attributes are independent of each other (hence naivety) and that all features are given an equal weighting (importance) [5]. Three main types (among other minor types) of Naïve Bayes Classifiers exist, but the one used was for the report was the Bernoulli classifier [4]. The Bernoulli Naïve Bayes classifier uses discrete data, where features are only in binary form, as well as a Bernoulli probability distribution. [5]. The program was initially made using the aforementioned classifier; however, due to poor performance, other classifiers were looked into and are discussed further in the report. The highest accuracy of 16.72% was generated using Bernoulli Naïve Bayes with 10-fold cross-validation along with the pre-processing steps, such as shuffling. Any number of folds beyond 10 did not seem to provide any noticeable benefit for accuracy and but still increased run-time overhead. The reason for this low accuracy is discussed in the Results section below.

To test other classification algorithms (like SVMs, Logistic Regression, Decision Trees, etc.), a pipeline was created to streamline the process. This allowed for the easy selection of various scikit-learn implementations through trial-and-error for the model-testing phase. The highest accuracy found during the testing phase was a combination of three unique models in a stacking approach, and is discussed in the succeeding Result section. During the training phase, stacking was the most effective model used as it provided the highest classifier confidence. Namely a stack combining linear SVC, ridge, and random forest classifiers, which will be discussed in the following paragraph. Bagging, a training method based on combining multiple variations of the prediction model (run independently from another to find a total average across all models) [5], was also tested; however, results were not consistent and provided slightly worse accuracies. For the validation phase, test accuracy validation found that after 10 folds, an insignificant improvement was made for accuracy, yet runtime exponentially increased.

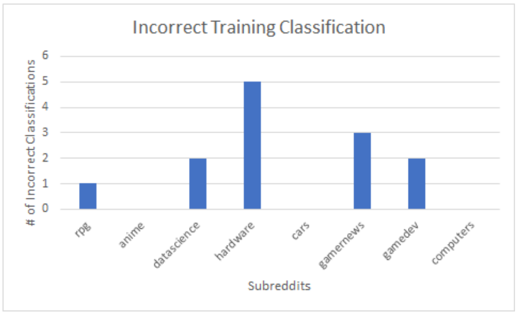
The linear SVC model is a modified support vector machine optimized for unsupervised learning [5]. Support vector machines are classifiers that can perform linear and nonlinear classification using what is called the kernel trick, which implicitly maps input to high-dimensional feature spaces [5]. A Ridge Classifier converts the input values into a negative or positive one and then treats the problem as a regression task [5]. Ridge Regression has the ability to increase performance for parameter estimation by simplifying the model, at the expense of also increasing the model's bias [1]. Random Forest is a type of meta-estimator that fits a set of decision tree classifiers to subsamples of the dataset [5]. It then averages results to improve accuracy score and reduce the common decision tree error of overfitting to training sets [3]. These classifiers were chosen because the underlying assumptions of each are drastically different. The biases of each algorithm are dissimilar and thus provide a much greater accuracy score. The final estimator was chosen to be linear SVC as it consistently generated the highest standalone probability of all the classifiers.

K-folds Cross Validation is a validation method that allows every observation to be used for both training and a one-time validation step. The K-fold method works by partitioning the dataset into k equal-sized portions, with a single subsample acting as the training set's validation data. The remaining k − 1 training subsamples are iterated k times, leaving each subsample to be used for validation only once [5]. The size of k is variable and generally set to an integer value, but was chosen to be 10 for this experiment [5]. A stratified version of KFolds is used in the stacking implementation as the folds preserve a percentage of samples for each class. This is done to slightly reduce bias, prevent class imbalance, and prevent over-representation [5].

5 Results

The accuracy of the Naïve Bayes classifier has an accuracy of 10.30% during the classification step. The reasoning for the low accuracy score is that the binarized vectorizer provides little information regarding the words in each document. As the data is purposely left unshuffled, there is occasionally a run-time warning that occurs. This is because classes can sometimes have a probability of 0, which becomes extremely prevalent as the data is not shuffled, and will calculate a value of log(0) (which is negative infinity). This heavily skews the prediction values, causing the classification to be questionably low. When shuffling, a probability of 13.37 is achieved. As mentioned above, a tf-idf weighting with a shuffled dataset provides much more data than a non-shuffled binarized vectorizer, at an accuracy rate of 16.72% and half the run-time. The algorithm's run-time was drastically slower, taking around 4x (2300 seconds) as long compared to the scikit-learn variant (603 seconds). This is because of the low-level optimizations implemented by scikit-learn, along with the efficient NumPy operations used when training.

The scikit-learn model provides a drastic improvement through the use of shuffling, tf-idf weighting, and the stacking of a linear SVC, ridge classifier, and random forest classifier. This stack was able to achieve an accuracy of 99.991% on the training dataset and a 91.48% accuracy on the Kaggle leaderboards. As our highest training classification featured only 1 incorrect training predictions, analysis on a slightly more incorrect permutation helped showcase which predictions were mislabelled. The majority were hardware and gamernews, and the reasoning behind such is discussed in the Discussion and Conclusion Section. This is visualized in the image below.



The graph represents an analysis of the incorrect training classifications, based on subreddit. It can be seen that the hardware and gamernews subreddits contain the highest number of errors.

6 Discussion and Conclusion

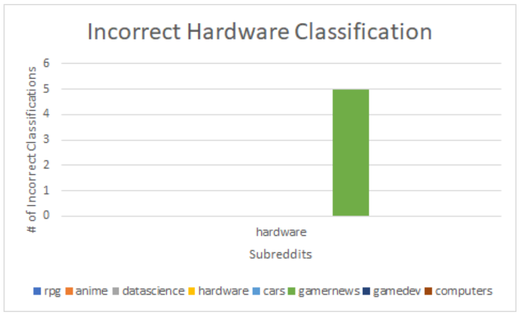
Throughout this entire experience, a few major takeaways seem to shine in a different light. To begin, by completing this project, the entire group possesses a greater understanding of natural language processing and the implications behind it. From tf-idf vectorization and stop word removal to lemmatization and maximum likelihood estimation, many classifiers have been looked at in-depth. Knowledge of text processing is increasingly important in the modern world, as it is being implemented in many industries. From search engines to language translation and content classification, natural language processing is now everywhere.

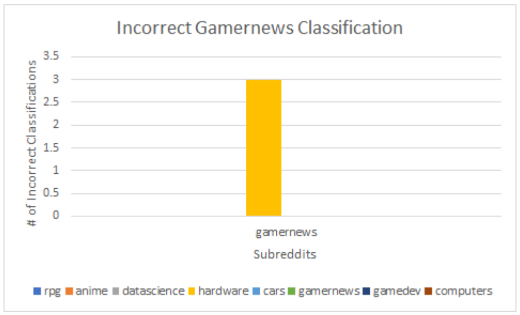
Not only did this provide a hands-on experience with modern NLP technologies, but it also allowed for exposure to cutting-edge libraries like scikit-learn and NumPy. Working with the entire scikit-learn pipeline has been incredibly informative and a great introduction to the machine learning world. The future of this project could go in many different directions, but as it stands, the two applications have great potential for useful application.

The first of which involves the analysis of popular Reddit posts. By creating a dataset consisting of the most viewed and up-voted posts, it would be possible to determine the commonalities between them. This would provide insight into what popular posts have in common both in terms of post context and user comments. This could be insightful for Reddit themselves, as it would provide a much better understanding of the platform's content. Not only that, but it would allow for much better business branding. While the first potential pathway revolves around commonality; the second delves deeper into community categorization.

Each community on Reddit has a set of guidelines that each post must conform to. Included in these is the rule that content must always be posted in the appropriate subreddit. Individual users are in charge of moderating each post and determining whether the appropriate content is posted to the corresponding subreddit. By creating a classification model that could determine what community a specific post belongs to, it could save countless hours for the site's moderators. As successful posts are easily identifiable through the up-vote system, this model could even generate new data over time. Moreover, as this would apply to thousands of subreddits, it is definitely a potential pathway to look at in the future.

Producing an accuracy score of 99.991% marks an incredible milestone as a higher value is quite unreasonable to achieve. The reason for this is that all of the data, both test and training sets, come from user data. This means that users may not always comment on something that is directly related to the subreddit assigned to it. An instance of this was seen in the hardware subreddit, as a user was referring to the Microsoft, Nintendo, and PlayStation game studios and spoke of content uncharacteristic of that subreddit. The same can be seen for a gamernews post that was incorrectly classified as hardware due to mentions of hardware limitations and specifications. As any reasonable algorithm would group hardware or gamernews content in the corresponding subreddit, the issue seems to stem from the collected data. Social media platforms - such as Reddit - lack the proper categorization that can be found in more research-based data sets. This is supported by the images below, which represent the subreddits that hardware and gamernews were incorrectly classified as.





7 Statements of Contribution

While each team member was responsible for creating components of the Bernoulli Naïve Bayes classifier as well as debugging, the Naïve Bayes and scikit-learn classifiers were primarily maintained by Michael Kazman, as well as Trevor Johns. Connor Stewart completed the majority of research, analysis, and reporting with slight adjustments made by Michael Kazman. The Data visualization and analysis was completed by Trevor Johns using Excel to generate the corresponding histograms.

References

[1] Felix Famoye. "Improving Efficiency by Shrinkage." In: *Technometrics* 41.2 (1999), p. 176. URL: `http://dblp.uni-trier.de/db/journals/technometrics/technometrics41.html#Famoye99`.

[2] Kendall Fortney. *Pre-Processing in Natural Language Machine Learning.* Nov. 2017. URL: `https://towardsdatascience.com/pre-processing-in-natural-language-machine-learning-898a84b8bd47`.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[4] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* USA: Cambridge University Press, 2008. Chap. Text classification and Naive Bayes, pp. 263–264. ISBN: 0521865719.

[5] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

8 Appendix

8.1 Appendix A – naive_bayes.py

```
import pandas as pd, numpy as np, sklearn, math, time
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import KFold
from sklearn.preprocessing import Normalizer

# NOTE: IF THIS CODE IS RUN LOCALLY, REMOVE THIS BLOCK AND PUT A train.csv FILE IN THE
    red↪ SAME DIRECTORY THE FILE IS IN
# import google drive (and mount it)
from google.colab import drive
drive.mount("/content/gdrive", force_remount=True)

# read in datasets ("file", epsilon)
train_data = pd.read_csv('/content/gdrive/My Drive/train.csv')
test_data = pd.read_csv('/content/gdrive/My Drive/test.csv')
# NOTE: IF THIS CODE IS RUN LOCALLY, REMOVE THIS BLOCK AND PUT A train.csv FILE IN THE
    red↪ SAME DIRECTORY THE FILE IS IN

#START NaiveBayesClassifier
class NaiveBayesClassifier:
  def __init__(self, labels):
    self.frequencies = {} # dictionary attribute
    self.max_features = 5000 # max number of words to categorize each post
    self.classes = ['rpg', 'anime', 'datascience', 'hardware', 'cars', 'gamernews', '
        red↪ gamedev', 'computers']
    self.class_theta = [] # theta values for each class (subreddit)
    self.feature_theta = [[ 0 ] * self.max_features] * len(self.classes) # theta values
        red↪ for each feature
    self.vectorizer = CountVectorizer(binary = True, max_features = self.max_features,
        red↪ stop_words = 'english') # converts csv into vectorizer matrix
    self.predictions = [] # list to store predictions

  ##START OF FIT SEGMENT
  def fit(self, posts, labels):
    # reset class theta (prior) values
    self.class_theta = []
    # turn the words into an array containing a vector of 1's / 0's (based on occurence
        red↪ of the top max_features words)
    words = self.vectorizer.fit_transform(raw_documents = posts, y = labels).toarray()
    # transform into single numpy matrix
    updated_data = np.c_[np.array(words), labels]
    # generate theta values for the classifier
    self.generate_theta(updated_data)
  #END fit

  def generate_theta(self, data):
    # go through the list of subreddits
    for (k, class_label) in enumerate(self.classes):
      # get the list of all rows that are part of the current subreddit (y = k)
      class_rows = data[data[:, -1] == class_label]
      # add the current class theta (current class posts / all posts)
      self.class_theta.append(len(class_rows) / len(data))
      # count the number of class rows
      num_class_rows = len(class_rows)
      # go through the set of features (m)
      for j in range(self.max_features):
        current_feature = class_rows[:, j]
        # sum up each of the columns
        sum = current_feature.sum()
        # j,k value (count of how many times the word was used in that class / all posts
            red↪ of that class)
        # additional laplace smoothing term is added here too
        self.feature_theta[k][j] = (sum + 1) / (num_class_rows + 2)
  ##END OF FIT SEGMENT

  ##START OF PREDICT SEGMENT
  def predict(self, posts):
    # reset predictions
    predictions = []
    # turn the words into an array containing a vector of 1's / 0's (based on occurence
        red↪ of the top max_features words)
    vectorized_posts = self.vectorizer.fit_transform(posts).toarray()
    # go through each post
    for post in vectorized_posts:
      class_probability = []
      for k in range(len(self.classes)):
        # summation term for each of the features probabilities
        feature_likelihood = 0
        # go through each of the features
        for j in range(self.max_features):
          # calculate and append likelihood term to the summation
          feature_theta = self.feature_theta[k][j]
          likelihood_term = (post[j] * np.log(feature_theta)) + ((1 - post[j]) * np.log(1
              red↪ - feature_theta))
          feature_likelihood += likelihood_term
        # calculate class probability
        class_probability.append(feature_likelihood + np.log(self.class_theta[k]))
      # determine highest probability class
      predicted_class = self.classes[np.argmax(class_probability)]
      # add class to prediction list
      predictions.append(predicted_class)
    return predictions

##START EXECUTION STEP
def main():
  # training step
  start = time.time()
  train_data = pd.read_csv('train.csv')
  train_rows = train_data.iloc[:, 0].to_numpy()
  train_labels = train_data.iloc[:, 1].to_numpy()
  trained_classifier = train(train_rows, train_labels)
  end = time.time()
  print('~ Training took ', end - start, ' seconds')

  # testing step
  start = time.time()
  test_data = pd.read_csv('test.csv')
  test_ids = test_data.iloc[:, 0].to_numpy()
  test_rows = test_data.iloc[:, 1].to_numpy()
  test(trained_classifier, test_ids, test_rows)
  end = time.time()
  print('~ Testing took ', end - start, ' seconds')
##END EXECUTION STEP

##START TRAINING STEP
def train(posts, labels):
  # parse the post and assigned subreddit into lists
  naive_bayes = NaiveBayesClassifier(labels)
  # sum of accuracies from all k folds
  accuracy_score = 0
  # setup k folds evaluation model
  num_folds = 10
  k_folds = KFold(n_splits = num_folds, random_state = None, shuffle = False)
  # iterate through k folds
  for (k, (train_folds, test_folds)) in enumerate(k_folds.split(posts)):
    # log the current fold
    print('~ Fold: ', k + 1)
    # separate train/test folds into corresponding rows and labels
    train_rows, train_labels = posts[train_folds], labels[train_folds]
    test_rows, test_labels = posts[test_folds], labels[test_folds]
    # update the internal weights of the classifier
    naive_bayes.fit(train_rows, train_labels)
    # return a list of predictions
    predictions = naive_bayes.predict(test_rows)
    # return an accuracy score of predictions vs actual labels
    evaluation_score = evaluation(predictions, test_labels)
    # add accuracy to summation
    accuracy_score += evaluation_score
  # calculate average accuracy
  average_accuracy = accuracy_score / num_folds
  print('~ Average accuracy is ', average_accuracy * 100, '%')

  # return the classifier for testing
  return naive_bayes

def evaluation(predictions, labels):
  # total values
  total = len(predictions)
  # uneven arrays
  if (total != len(labels)):
      print('~ ERROR: Evaluation list mismatch')
      return -1
  # empty arrays
  if (total == 0):
      print('~ ERROR: Evaluation list empty')
      return 1
  # set evaluation total to 0
  correct_count = 0
  # calculate true error
  for i in range(total):
    if (predictions[i] == labels[i]):
        correct_count += 1
  # define accuracy
  accuracy = correct_count / total
  # return rounded accuracy
  return (accuracy * 100) / 100.0
#END evaluation
##END TRAINING STEP

##START TEST STEP
def test(classifier, ids, posts):
  # generate list of predictions
  predictions = classifier.predict(posts)
  # export list to csv
  export_to_csv(ids, predictions)

def export_to_csv(ids, predictions):
  # containers for csv
  id_list = []
  prediction_list = []
  # iterate through the predictions
  for (index, prediction) in enumerate(predictions):
    # append id and prediction to list
    id_list.append(ids[index])
    prediction_list.append(prediction)
  # create in pandas dataframe
  dataframe = pd.DataFrame({ 'id': id_list, 'subreddit': prediction_list }, columns = ['
      red↪ id', 'subreddit'])
  # convert to csv
  output_file = 'test_results.csv'
  dataframe.to_csv(output_file, index = False)
  print('~ Predictions exported to ', output_file)
#END export_to_csv
##END TEST STEP

main()
```

8.2 Appendix B – scikit.py

```
import pandas as pd, numpy as np, sklearn, math, time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC, LinearSVC
from sklearn.linear_model import RidgeClassifier
from sklearn.ensemble import RandomForestClassifier, StackingClassifier,
    red↪ BaggingClassifier

# NOTE: IF THIS CODE IS RUN LOCALLY, REMOVE THIS BLOCK AND PUT A train.csv FILE IN THE
    red↪ SAME DIRECTORY THE FILE IS IN
# import google drive (and mount it)
from google.colab import drive
drive.mount("/content/gdrive", force_remount=True)

# read in datasets ("file", epsilon)
train_data = pd.read_csv('/content/gdrive/My Drive/train.csv')
test_data = pd.read_csv('/content/gdrive/My Drive/test.csv')
# NOTE: IF THIS CODE IS RUN LOCALLY, REMOVE THIS BLOCK AND PUT A train.csv FILE IN THE
    red↪ SAME DIRECTORY THE FILE IS IN

##START EXECUTION STEP
def main():
  # training step
  start = time.time()
  #### IF THIS CODE IS RUN LOCALLY UNCOMMENT THIS BLOCK
  # train_data = pd.read_csv('train.csv')
  #### IF THIS CODE IS RUN LOCALLY UNCOMMENT THIS BLOCK
  train_rows = train_data.iloc[:, 0].to_numpy()
  train_labels = train_data.iloc[:, 1].to_numpy()
  (classifiers, vectorizer) = train(train_rows, train_labels)
  end = time.time()
  print('˜ Training took ', end - start, ' seconds')

  # testing step
  start = time.time()
  #### IF THIS CODE IS RUN LOCALLY UNCOMMENT THIS BLOCK
  # test_data = pd.read_csv('test.csv')
  #### IF THIS CODE IS RUN LOCALLY UNCOMMENT THIS BLOCK
  test_ids = test_data.iloc[:, 0].to_numpy()
  test_rows = test_data.iloc[:, 1].to_numpy()
  test(test_ids, test_rows, classifiers, vectorizer)
  end = time.time()
  print('˜ Testing took ', end - start, ' seconds')
##END EXECUTION STEP

##START TRAINING STEP
def train(posts, labels):
  # start timer
  start = time.time()
  # initialize vectorizer
  vectorizer = TfidfVectorizer(stop_words = 'english', max_df = 1.25)
  # vectorize dataset
  vectorized_posts = vectorizer.fit_transform(posts)
  # classifiers to be stacked
  classifiers = [
    ('1', LinearSVC()),
    ('2', RidgeClassifier()),
    ('3', RandomForestClassifier())
  ]
  # number of folds for k-folds
  num_folds = 10
  # initialize stack
  classifier = StackingClassifier(cv = num_folds, estimators = classifiers)
  #classifier = BaggingClassifier(base_estimator = RandomForestClassifier(), n_estimators
      red↪  = 30)
  classifier.fit(vectorized_posts, labels)
  # fit and predict model
  classifier.fit_transform(vectorized_posts, labels)
  accuracy = classifier.score(vectorized_posts, labels)
  # end timer
  end = time.time()
  # print accuracy
  print('Total Accuracy: ', accuracy * 100, '%, and took ', str(round(end - start, 2)), '
      red↪  seconds')
  # iterate predictions for analysis (on miscalculated rows)
  prediction_probabilities = classifier.predict_proba(vectorized_posts)
  predictions = []
  for probabilities in prediction_probabilities:
    predictions.append(classifier.classes_[np.argmax(probabilities)])
  # export for training data analysis
  export_to_csv(predictions, 'training_predictions.csv')
  # return classifier and vectorizer (for testing)
  return (classifier, vectorizer)

##START TEST STEP
def test(ids, posts, classifier, vectorizer):
  # transform data
  vectorized_posts = vectorizer.transform(posts)
  # predict posts
  predictions = classifier.predict(vectorized_posts)
  # export list to csv
  export_to_csv(predictions, 'test_results.csv')

def export_to_csv(predictions, filename):
  # containers for csv
  id_list = []
  prediction_list = []
  # iterate through the predictions
  for (index, prediction) in enumerate(predictions):
    # append id and prediction to list
    id_list.append(index)
    prediction_list.append(prediction)
  # create in pandas dataframe
  dataframe = pd.DataFrame({ 'id': id_list, 'subreddit': prediction_list }, columns = ['
      red↪ id', 'subreddit'])
  # convert to csv
  dataframe.to_csv(filename, index = False)
  print('~ Predictions exported to ', filename)
#END export_to_csv
##END TEST STEP

main()
```