Chapter 1

INTRODUCTION

Human languages are difficult to be understood by machines as well as by us. In this developing era, one human language can mean some things. Also, in this era, a lot of data has been emerged. The growth of data in recent years has been impressive, with the amount of data generated continuing to grow exponentially [7].

"Data really powers everything that we do." – Jeff Weiner [7].

The data that are growing can be in text documents, images, video files, audio, etc. The data will grow year by year. According to Symantec's 2013 State of Information Report, data is growing at a rate of 60% to 70% annually [3].

According to industry estimates, only 21% of the available data is present in structured form. Data is being generated as we speak, as we tweet, etc. Majority of this data exists in the textual form, which is highly unstructured in nature [8].

So, to handle the unstructured text data, NLP comes into the play. And its branch text classification helps a lot in classifying the text data. Text classification in NLP involves categorizing and assigning predefined labels or categories to text documents, sentences, or phrases based on their content. It automatically aims to determine the class or category to which a piece of text belongs [16].

The benefits of supervised and unsupervised learning are known. Unsupervised learning can deal with data without class labels. However, in supervised learning labeled data is dealt with and labelling the unlabeled data is costly. So, in this proposed project, the gathered unlabeled data are labeled using similarity between the labeled and unlabeled documents. It saves time for people in labelling.

This project, Label The Unlabeled Data Using Supervised Learning, the will reduce the human efforts with labelling the unlabeled dataset.

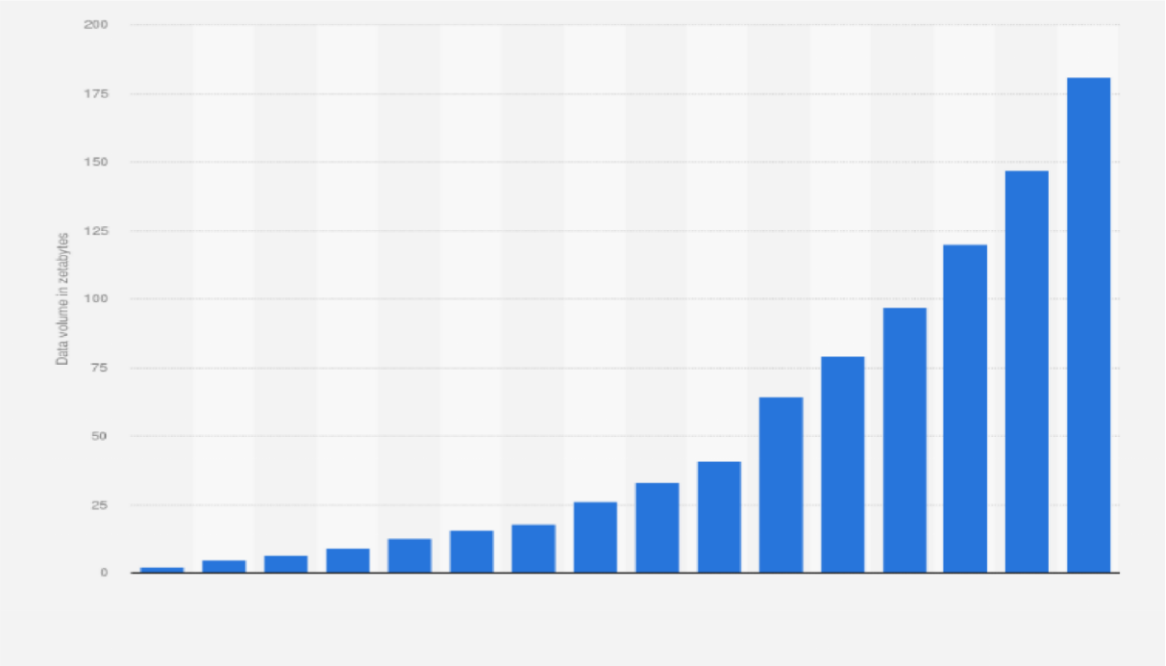
1.1 Data Growth:

Data growth refers to the constant increase in the amount of data being produced by modern technological society as well as the increase in the amount of data an enterprise must store [3].

Data has been growing exponentially for the last few decades. According to the Data Services Market Report, the main drivers of growth are remote work, the digitization of existing processes, and the growing industrial sector using digital technologies, a rise in number of SMEs adopting digital technologies, growing used of Over-the-Top (OTT) services (a method of delivering film and television directly via the internet, without the need for traditional cable or satellite broadcasting), and the development of data-generating and data-hungry technologies, such as IoT or Machine Learning (ML).

The growth of data in recent years has been impressive, with the amount of data generated continuing to grow exponentially. The Internet and the rapid growth of Internet users over the years is one reason for this, as are the data created by large enterprises, tech companies, ecommerce platforms, health and scientific industries, governmental and non-governmental organizations. The tech giants, including Meta, Amazon, Google and Microsoft are major contributors to the growth of data.

The following chart is the growth of data from 2010 to 2025.



(Source: statista) Fig-1: Growth of Data from 2010 to 2025

Since 2010 data has been growing (according to the chart above), and by 2030 the amount of data generated (grown) worldwide would increase significantly. According to current trends, experts estimate the data growth would reach 660 zettabytes or more.

1.2 Text Classification:

Text or document classification or categorization is the process of assigning text documents into one or more classes or categories, assuming that we have a predefined set of classes. Documents here are textual documents, and each document can contain a sentence or even a paragraph of words. A text classification system would successfully be able to classify each document to its correct class (es) based on inherent properties of the document. The textual data involved can be anything ranging from a phrase, sentence, or a complete document with paragraphs of text, which can be obtained from corpora, blog or anywhere form the Web.

Text classification in NLP (Natural Language Processing) involves categorizing and assigning predefined labels or categories to text documents, sentences, or phrases based on their contents. Text classification aims to automatically determine the class or category to which a piece of text belongs. It's a fundamental task in NLP with numerous practical applications, including sentiment analysis, spam detection, topic labelling, language identification and more. Text classification algorithms analyze the features and patterns within the text to make accurate predictions about its category, enabling machines to organize, filter, and understand large volumes of textual data [16].

1.3 Problem Statement:

Machines cannot understand our human languages. For example, English words can't be understood by machines, so how can we make machine understand those words? And how can we label the raw dataset? And how can we make machines those words to be classified or categorized?

The main problem is that when we are working with text data, first they are unstructured which is difficult to analyze, organize and extract meaningful insights. There will be information overload, lack of structure, inefficient search and limited automation.

Text (or document) classification or categorization comes into play. In text classification, the unstructured text data are converted into structured text data which makes easier to classify or organize the text data.

1.4 Motivation:

First, ‘how can we know this statement belongs to this class?’. In simple words, I got confused and rose a question like, for example in movie review, how can I know a review is positive or love review? And same goes to comments on various social media; Twitter, Facebook, Instagram, etc. like the question above. Now, we have NLP to do this by using many techniques like Machine Learning techniques, Deep Learning techniques, etc.

Using NLP, we can categorize or classify the text data. Let’s say, we have an incoming text data to a model or machine and we want to know to what this text data belongs. We do classification here. Even in the reviews, we can analyze what comments are negative comments and what comments are not.

In performing above such classification, when we gather raw data, we can name or give the class of each document but if the number of documents is very large, manual labelling will result into time consuming. We need a way to label the unlabeled dataset. So, this project, Label The Unlabeled Data Using Supervised Learning, is the right idea instead of training many algorithms

1.5 Contribution:

In order to classify some text data to the category or class it belongs, NLP is used with ML techniques. Here, in this project, the main focus is the labelling the unlabeled document. When we have labeled and unlabeled datasets, we can use the labeled dataset to predict or generate the class labels of the documents in the unlabeled dataset, by finding similarity.

And, to classify a new text data to which category it’s belonged is also one of main goals in this project. For any new text data, we are going to classify. Machines don’t understand the text data, so we convert the text data into numbers and feed them to the trained and test model (algorithm) to classify or predict to which category the text data belongs.

Chapter 2

LITERATURE SURVEY

Text data are easily available and can be collected from different data repository sites. Manual analysis simply cannot keep pace with the fast accumulation of massive data. So, machine learning techniques such as feature engineering, classification, regression, clustering, etc. help analyze the text data.

Base on the nature of data gathered, what type of machine learning method would be useful for analyzing the data. If the gathered data is fully labeled, then we can use supervised learning, if not we can use unsupervised learning, if the gathered data has labels and also the data don't have labels semi-supervised learning can be used or hybrid approach (supervised and unsupervised).

All users want to have their documents in a more systematic and secured way. Assume a situation. We have huge collections of books. It may contain novels, storybooks, and fictions, etc. Suppose someone enquires a book on History, it is quite difficult for us to find it in the midst of all books. If we manually go for searching, it may take several hours, may be days. If we can categorize the books in different categories with respect to some criteria it would have been more efficient to search and more secured too [3]. Text classification in NLP (Natural Language Processing) involves categorizing and assigning predefined labels or categories to text documents, sentences, or phrases based on their contents [16].

Unlabeled documents vastly outnumber labeled the documents in text classification. For this reason, semi-supervised learning is well suited to the task. Representing text as a combination of unigrams and bigrams in semi-supervised learning using two algorithms Co-Training and Self-Training [11].

Self-Training is a wrapper semi-supervised algorithm that makes use of a supervised learning algorithm that trains itself. And Co-Training algorithm is the multi-view counterpart of Self-Training. Interest in multi-view algorithms comes from the fact that they take advantage of multiple representation [11].

Text categorization is a problem which can be addressed by a semi-supervised learning classifier, since the annotation process is costly and ponderous. A novel approach for semi-supervised learning based on WiSARD classifier (SSW), and compares it to other already established mechanisms (S3VM and NB-EM), the novel approach showed to be up to fifty times faster than S3VM and NB-EM. To adapt WiSARD to work on a semi-supervised fashion, the use of a predicting confidence to decide which unlabeled pattern will be trained. The use of confidence metric: c=1-r1/r2, r1 is the best result from a discriminator and r2 is the second best result [12].

The manual data labeling method is slow and expensive, it also cannot cope with enormous amount of data. A semi-supervised deep neural network is proposed, to reduce the effort and time needed in data labelling as it uses a combination of small amount of labeled data and a large pool of unlabeled data for model training. Supervised learning produces a good performance model however required a fully labeled corpus. On the other hand, unsupervised learning needs no labeled dataset but gives comparatively poor performance. Semi-supervised learning makes use of a combination of labeled and unlabeled data [2, 11].

The goal of text categorization is to classify documents into a certain number of predefined categories [15]. It is so difficult to create the labeled training documents and not easy to manually categorize them for creating training documents [11, 15, 13]. The proposed method divides the documents into sentences, and categorizes each sentence using keyword lists of each category and sentence similarity measure [15].

Chapter 3

METHODOLOGY

The most crucial and first step is the collection or gathering the data. Then preprocessing the gathered data. Preprocessing is a must to reduce noise and unwanted things in our dataset. There are many steps or methods to preprocess a dataset. After preprocessing, the text data must convert into numerical values, so feature extraction is needed here. Applying feature extraction is a must thing.

The feature engineering is the process in which the raw dataset is transformed into flat features which can be used in a machine learning model. This includes the process of creating new features from the existing data [18]. Feature extraction is a fundamental task that involves converting raw text data into a format that can be easily processed by machine learning algorithms [17].

First the dataset is split into train (80%) and test (20%), only the train subset is taken. The mean of documents in every class in the training set is calculated, which is a vector form. The documents in the test subset is also in vector form, which are same in dimension.

The test set has its own labels, however we didn't take it. Using a similarity function to assign the labels to each document in the test set. So, cosine similarity is used and find the similarity between the mean of each class in training set and each document in test set. If the similarity between a specific class and the vector representation of a document in test set is higher, then the document in the test set gets the label of that specific class. Means, maximum the similarity is the class label of the document. After assigning respective labels to the documents of testing set, classification is performed.

Again, the dataset is split into train (30%) and test (70%), like above only the training set is taken, and find mean of each class, and find similarity between the mean and each document in the test set. And maximum the similarity is the class label of the document in the test set. Again, carried out classification.

Both the experiments give similar accuracy, so the methodology works well.

Second, this involves two datasets, one labelled dataset which has 25 categories and one unlabeled dataset. The mean of documents of each class in the labelled dataset is calculated. Then, find similarity between the mean and each document of the unlabeled dataset, and assign class label with high similarity. The unlabeled dataset is now labelled with this method.

Then, the two datasets (in order to perform classification) are combined, and perform classification which gives good accuracy.

Again, like the above, two datasets are taken, one labelled dataset which has 25 categories or classes or groups and one unlabeled dataset. Finding the mean of each class of the labelled dataset and find similarity between the mean and each document of the unlabeled dataset. Maximum the similarity is the class label of the document of unlabeled dataset. So, the class labels of the documents in unlabeled dataset are now assigned.

Both the datasets are now having class labels, then the two datasets are combined and perform classification which also gives good accuracy.

Third, this approach is ongoing development, the datasets are same as before like just above, one labelled which has 25 classes and unlabeled dataset. On unlabeled dataset, one of the methods of unsupervised learning which is clustering (k-means) is applied, which has 25 clusters. The number of classes in the labelled dataset and number of clusters in the unlabeled dataset are same. Now, for each cluster, means of each cluster and also for each class are calculated. Then, find the similarity between them and maximum the similarity between them, is the class label of the cluster. So, each cluster has class labels which is like labelled dataset. After this, we can perform classification by combining the datasets and splitting into train and test sets.

The following is the conceptual figure of the proposed project:

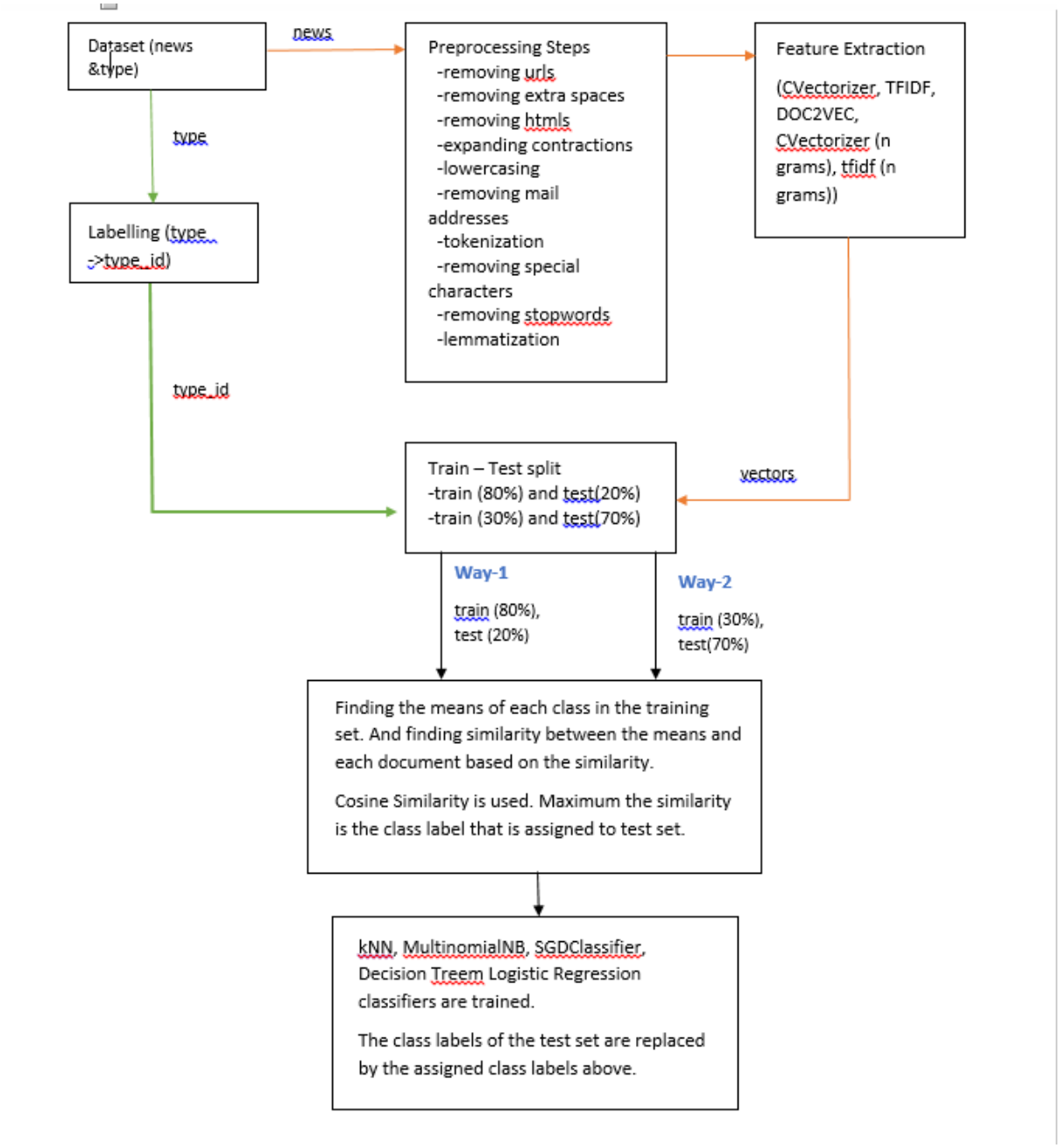


Fig-2.1: Conceptual Framework of the proposed model

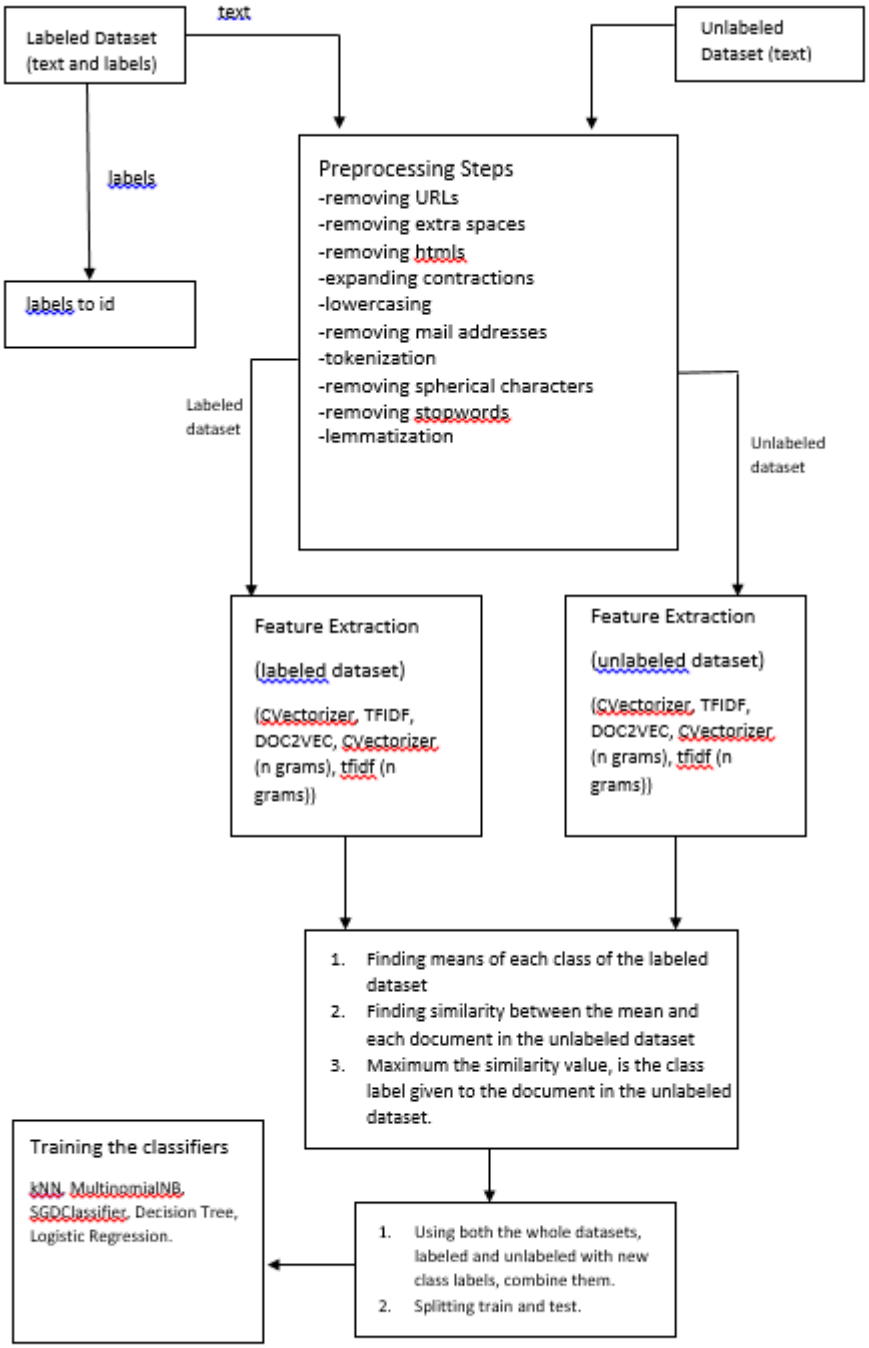


Fig-2.2 : Conceptual Framework of the proposed model

3.1 Dataset:

The dataset 20newsgroups used in this project is from UCI Repository Dataset, created by Tom Mitchell.

The data type of the dataset is text. This dataset consists of 20000 messages taken from 20 newsgroups. There are 20 groups (folders) in the dataset. One thousand Usenet articles were taken from each of the 20 newsgroups (folder). Each newsgroup is stored in a subdirectory, with each article stored as a separate file.

(Source: https://archive.ics.uci.edu/dataset/113/twenty+newsgroups)

The BBC five news groups dataset is from Kaggle. The dataset comprises of classes - “Business, Entertainment, Politics, Sport, Tech”.

And another dataset is from Kaggle. This dataset is collected by the uploaded by Kishan Yadav on Kaggle.com, which full credit is given to him. According to him, this data is collected on daily basis from inshorts news webapp.

(Source: https://www.kaggle.com/datasets/kishanyadav/inshort-news)

3.2 Preprocessing:

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. Here in textual data, data preprocessing is also known as text preprocessing (NLP).

Text data can come from many sources like websites, social media, books, etc. The sources may have many unwanted things and noises. And unready or unreliable to train models. So, we do text preprocessing. Text preprocessing aims to remove or reduce the noise or unwanted things and making the dataset ready or reliable to use in training model.

In NLP, there are many steps involve in text preprocessing. Let’s discuss the ones I included in this project.

1. Removing URLs:

   When we have text dataset, there may be many URLs (Uniform Resource Locaters) pointed or linked to some websites. We don’t need these in training the models, so we remove them.

2. Removing Extra Space:

   There may be many extra white spaces, so in order to make our dataset look nice, we can remove those extra spaces.

3. Removing htmls:

   The htmls tag are removed here, because they don't mean good to our models.

4. Expanding Contractions:

   Contractions are English words (I use English in my project). But in order to train models we don't need these contractions, so we expand common contractions.

5. Standardization:

   In here, I convert every word to lowercase to make training the model easier.

6. Removing Mails addresses:

   In the dataset I use, there are many mail addresses. And we don't need these in training the models, so I remove every one of them.

7. Tokenization:

   Tokenization is a technique that involves dividing a sentence or phrase into smaller units known as tokens. These tokens can encompass words, dates, punctuations marks, or even fragments of words.

8. Removing Special Characters:

   We don't special characters to train a model. Special characters include punctuations, ASCII code, etc. So, it's better to remove these characters.

9. Removing Stopwords:

   I used stopwords in NLTK library. We don't need these stop words to be included in training the model, so we need to remove them from the dataset.

10. Lemmatization:

    Lemmatization is a process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. Lemmatization is preferred over stemming because it does morphological analysis of the words.

3.3 Feature Engineering:

The feature engineering is the process in which the raw dataset is transformed into flat features which can be used in a machine learning model. This includes the process of creating new features from the existing data [18].

Feature engineering is one of the most important steps to be followed for a better understanding of the context of what we are dealing with. After the initial text is cleaned (preprocessed), we need to transform it into its features to be used for modeling. Document data is not computable so it must be transformed into numerical data such as vectors.

Machine learning algorithms do not understand the text data, so we need to transform the text to numerical forms called vectors. There is a step called Text Vectorization or Text Representation in NLP which transform the text to vectors [5].

There are some numbers of feature extraction (text vectorization) techniques. However, in this project, CountVectorizer, CountVectorizer with n-grams, TFIDF, TFIDF with n-grams and Doc2Vec are used.

3.3.1 TF-IDF (Term Frequency and Inverse Document Frequency):

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

Term Frequency (TF):

The number of times a word appears in a document is divided by the total number of words in that document, $0<TF<1$.

$$TF_{ij} = \frac{\text{Number of times term i appears in the document}}{\text{Total number of terms in the document j}}$$

Inverse Document Frequency (IDF):

The logarithm of the number of documents in the corpus is divided by the number of documents where the specific term appears. In sklearn use log(N/ni)+1 formula.

$$IDF_i = \log(\text{Total number of documents} / \text{Number of documents with term i in it})$$

In TFIDF, we take a log to calculate IDF because it we have a very rare word, the IDF value without a log is very high, and then we have to calculate TF*IDF, at this time IDF value will dominate the TF value because TF lies in 0 and 1. That means we normalize the IDF value using a log.

(from sklearn.feature_extraction.text import TfidfVectorizer) [10]

3.3.1.1 TF-IDF (n grams):

By Wikipedia, "N-gram is a continuous sequence of N items from a given sample of text or speech" [6]. Sklearn provides tfidf(ngram_range) as attribute to create n-grams space. So, I give ngram_range as (1, 3), means I am using unigram, bigram and trigram in just one feature extraction technique. One recommendation here that when we are using this kind of range, this will create a very big space so, the attribute in tfidf max_features should be some value smaller, like 5000. Because the memory size is very large if we are dealing with this kind of range. However, I gave max_features as 10000.

Unigram: this is when the value of n is one, means only one word.

Bigram: this is when the value of n is two, means two consecutive words.

Trigram: this is when the value of n is three, means three consecutive words

3.3.2 Count Vectorizer:

CountVectorizer is a class in the scikit-learn library of Python that is used for converting a collection of text documents into a matrix of token counts. It creates a bag of words representation of the text corpus, where each document is represented as a vector of term frequencies. The countvectorizer class provides various options for preprocessing the text data, such as tokenization, removing stop words, and stemming. It also allows for the specification of the maximum vocabulary size and minimum document frequency required for a term to be included in the vocabulary. The resulting matrix can be used as input to various machine-learning models for tasks

such as text classification and clustering. (from sklearn.feature_extraction.text import CountVectorizer) [10]

3.3.2.1 Count Vectorizer with n-grams:

Also, as in the TFIDF, sklearn provides ngram_range in CountVectorizer. The range is same as before in TFIDF and also the value of max_features. So, unigram, bigrams and trigrams are used.

3.4 DOC2VEC:

DOC2VEC or Doc2Vec is a neural network-based approach that learns the distributed representation of documents. It is an unsupervised learning technique that maps each document to a fixed-length vector in a high-dimensional space.

Doc2Vec is an extension of Word2Vec that can create vectors for entire documents, capturing semantic implementation that goes beyond individual words. The gensim library provides an efficient implementation of Doc2Vec.

First, we need to install gensim, pip install gensim. Then, import the library and also the Doc2Vec.

```
import gensim
from gensim.models.doc2vec import Doc2Vec
```

3.5 Labelling:

The dataset which comprises of 25 classes is read. And then, after feature extraction, the dataset is split into train (80%) and test (20%). Using the class labels in the training set, the mean of documents in a particular class label is calculated and find similarity between the means and document vectors of the documents in the test set. This means, the class labels in the test set are not used. If the similarity finds the highest, then the class label of the class which involves in the similarity comparison is assigned to the document which involves in the comparison.

The assigned labels or predicted labels are used find how accurately those are by calculating accuracy between the predicted labels and actual labels. And the predicted labels are used to train the algorithms.

After this, again the dataset is split into train (30%) and test (70%). Then previous steps are performed. This also works well.

So, after this, the above labeled dataset and another unlabeled dataset are used. With the calculated means of the classes of the labeled dataset and document vectors of the documents in the unlabeled dataset, the similarity is calculated and the class labels of the documents of the unlabeled dataset are assigned. Maximum the similarity, the class label is.

3.6 Another approach:

The unlabeled dataset is used to perform clustering (k-means). Then the labeled dataset is used for the labelling of the documents in the unlabeled dataset. The mean of each class of the labeled document is calculated and also the mean of each cluster, then find the similarity between them. So, the maximum the similarity between them, is the class label of each cluster.

However, this approach is ongoing development.

3.7 Algorithm:

The algorithm used in this project is given below:

1. Find vector representation of the text data,
2. Find mean vectors of the documents that belong to a particular class,
3. Find similarity between the mean vectors and vector representations of each document in the unlabeled dataset,
4. Assign class labels of the class that is involved in finding the similarity to the document that is involved in finding the similarity, if the similarity is maximum.
5. Repeat the step 3 and 4 until all the documents in the unlabeled dataset are labeled.

3.8 Classification:

Classification is a supervised learning technique that is used to identify the category of new observations on the basis of training data. In classification, a program leans from the given dataset or observations and then classifies new observation into number of categories. The main objective of classification is to build a model that can accurately assign a label or category or class to a new observation based on its features.

Supervised Learning is a type of Machine Learning, where we have input variables (X) and an output variable (Y) and we use an algorithm to learn the mapping function from the input to the output Y=f(X). The goal is to approximate the mapping function so well that when we have new input data (x), we can predict the output variables (Y) for that data.

In classification, we have two types: binary and multi-class classifier.

Binary Classification:

If the classification problem has only two possible outcomes (classes or labels or categories, etc.), then it is called as binary classification.

Multi-Class Classification:

If a classification problem has more than two classes or labels, etc., then it is called as multi-class classification.

The project I give is the multi-class classification having 20 (twenty) classes or labels or categories.

The classifiers I used in this project are six in numbers. Let's define one by one:

3.8.1 Classifiers:

A classifier is an algorithm that automatically categorizes data into one or more categories or classes. Targets, labels and categories are all terms used to describe classes.

1. kNN:

   kNN is one of the simplest ML algorithms based on supervised learning technique. It assumes the similarity between the new case/date and available cases and put the new case into the category that is most similar to the available categories. When new data appears then it can be easily classified into a well suite category by using

kNN algorithm. It is a non-parametric algorithm, which means it does not make any assumption on underlying data.

In kNN, k means the number of the nearest neighbors (we can select any number depending on the model). Calculate any distance producing method (Euclidean or Manhattan or any other) to find the similarity. The minimum distance between the new data point and some neighbors of a category is the solution, means the new data point will be fed or kept in that category.

I used kNN which is already available in sklearn library. And the value is k is 5. (from sklearn.neighbors import KNeighborsClassifier)

2. Multinomial Naïve Bayes:

   Naïve Bayes algorithm or classifier is a supervised learning algorithm which is based on Bayes' Theorem of probability. It is probabilistic classifier, which means it predicts on the basis of the probability of an object.

   In this project, I used Multinomial Naïve Bayes. It is a very popular and efficient ML algorithm that is based in Bayes' theorem. It is commonly used in text classification of multiple classes.

   Multinomial Naïve Bayes is a probabilistic classifier to calculate the probability distribution of text data, which makes it well-suited for data with features that represent discrete frequencies or counts of events in various NLP tasks. (from sklearn.naive_bayes import MultinomialNB)

3. SGD Classifier:

   SGD (Stochastic Gradient Descent) is one of the popular optimization methods in Deep Learning (DL) and ML. Large datasets and complicated models benefit greatly from its training. To minimize a loss function, SGD updates model parameters iteratively. It differentiates itself as stochastic by employing minibatches, or random subsets, of the training data in each iteration, which introduces a degree of randomness while maximizing computational efficiency. By accelerating convergence, this randomness can aid in escaping local minima. Modern machine learning algorithms rely heavily on SGD because, despite its simplicity, it may be quite effective when combined with regularization strategies and suitable learning rate schedules.

SGD Classifier has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification. It is merely an optimization technique and does not correspond to a specific family of machine learning models.

I used SGD Classifier given in sklearn library, which I can use in my project. It uses one-vs-rest approach. And default loss function is hinge loss. (from sklearn.linear_model import SGDClassifier)

4. Logistic Regression:

Logistic Regression is a statistical method used mainly for binary classification the models the probability of a binary outcome based on one or more predictor variables. It is a fundamental technique in machine learning and statistics, commonly used when the dependent variable is categorical. This can also be used for multi-class classification by using techniques like One-vs-Rest and Multinomial (Softmax) Logistic Regression.

I used the logistic regression from scikit-learn which provides a straightforward implementation which uses OvR by default. (from sklearn.linear_model import LogisticRegression)

3.9 Evaluation Matric:

Evaluation metrics are tied to ML tasks. By looking at this matric we can evaluate the models' performance. In ML, we have confusion matrix.

A confusion matrix gives a comparison between actual and predicted values. It is used for the optimization of ML models. The confusion matrix is an N x N matrix, where N is the number of classes or outputs.

In this, we have; TP = True Positive, TN= True Negative, FP= False Positive, FN= False Negative.

TP = true positive value is where the actual value and predicted value are the same.

TN = true negative value for a class will be the sum of the values of all columns and rows except the values of that class we are calculating the values for.

FP = false positive value for a class will be the sum of values of the corresponding column except for the TP value.

FN = false negative value for a class will be the sum of values of corresponding rows except for the TP value.

The confusion matrix allows us to measure Recall and Precision, which, along with Accuracy and the AUC-ROC curve, are the metrics used to measure the performance of ML models.

Chapter 4

TOOLS

I used Anaconda navigator which has jupyter notebook in building this project, python programming language and also some libraries are used. And the vector space is very large, I took help from Google Colab.

4.1 Python Programming Language:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

4.1.1 Python Over Other Programming Languages:

Let's describe some advantages of Python over other programming language.

- Presence of third-party modules:

  Python has a rich ecosystem of third-party modules and libraries that extend its functionality for various tasks.

- Open source and large active community base:

  Python is open source, and it has a large and active community that contributes to its development and provides support.

- Versatile, easy to read and write:

  Python is known for its simplicity and readability, making it an excellent choice for both beginners and experienced programmers.

4.1.2 Python in Machine Learning:

Machine Learning (ML) is the field of study that gives computers the capability to learn without being explicitly programmed.

Python has a crucial role in ML because Python provides libraries like NumPy, Pandas, ScikitLern, Tensorflow, and Keras.

4.1.3 Python in Natural Language Processing:

Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI). This is widely used technology for personal assistants that are used in various business fields/areas. This technology works on the speech provided by the user breaks it down for proper understanding and processes it accordingly. NLP is an upcoming field where already many transitions such as compatibility with smart devices, and interactive talks with a human have been made possible.

Python is a popular programming language for NLP tasks because it is easy to learn and use, and it has a large community of developers who contribute to its libraries and tools. Python has a number of powerful libraries for NLP tasks, such as NLTK, spaCy, and TextBlob. These libraries provide a variety of functions and tools for tasks such as tokenization, stemming, lemmatization, POS tagging, NER (Named Entity Recognition), and sentiment analysis.

4.2 Libraries:

A Python library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs.

Below are libraries I used in this project:

NLTK (Natural Language Toolkit):

NLTK is the main library for building Python projects to work with human language data. It gives simple to-utilize interfaces to more than 50 corpora and lexical assets like WordNet, alongside a set-up of text preprocessing libraries for tagging, parsing, classification, stemming, tokenization, and semantic conversation discussion.

Scikit-learn:

It is one of the greater Python libraries for NLP and is most used among data scientists for NLP tasks. It provides a large number of algorithms to build machine learning models.

Gensim:

Gensim is a robust Python library designed for topic modelling, document indexing and similarity retrieval with large corpora. Developed by Radim Rehurek, Gensim is specifically tailored for handling large-scale text collections efficiently.

Other:

In this project, I used pandas, string, re. Pandas for creating data frame, string for some preprocessing step, and re (Regular Expression) to remove or replace some symbols or words.

Then I used pickle and streamlit to make prediction interface for the new text data.

Chapter 5

RESULT AND DISCUSSION

5.1 Result:

First, let's add the results of working with tfidf:

(train – 80% and test – 20%)

(train – 30% and test – 70%)

When train is 80% and test is 20%

The accuracy of actual test labels and predicted or assigned is:

```
accuracy = accuracy_score(y_test, test_class_predictions)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.8233131487889274
```

Accuracy of the classifiers:

	precision	recall	f1-score	support
0	0.68	0.85	0.76	177
1	0.66	0.65	0.65	224
2	0.56	0.63	0.59	203
3	0.72	0.72	0.72	201
4	0.66	0.74	0.70	175
5	0.72	0.71	0.72	185
6	0.71	0.48	0.57	232
7	0.79	0.83	0.81	203
8	0.86	0.93	0.90	183
9	0.85	0.88	0.87	189
10	0.86	0.95	0.90	197
11	0.82	0.91	0.87	186
12	0.73	0.58	0.65	245
13	0.88	0.86	0.87	185
14	0.84	0.89	0.86	182
15	0.90	0.79	0.84	230
16	0.83	0.68	0.75	243
17	0.86	0.96	0.91	187
18	0.65	0.58	0.61	213
19	0.58	0.71	0.64	157
20	0.87	0.89	0.88	194
21	0.89	0.89	0.89	151
22	0.94	0.94	0.94	102
23	0.92	0.84	0.88	95
24	0.96	0.88	0.92	85
accuracy			0.78	4624
macro avg	0.79	0.79	0.79	4624
weighted avg	0.78	0.78	0.77	4624

kNN with predicted labels.

Below is with actual labels:

	precision	recall	f1-score	support
0	0.69	0.77	0.73	200
1	0.69	0.76	0.72	200
2	0.64	0.72	0.68	200
3	0.71	0.72	0.72	200
4	0.75	0.74	0.74	200
5	0.83	0.76	0.79	200
6	0.72	0.56	0.63	200
7	0.82	0.89	0.85	200
8	0.91	0.91	0.91	200
9	0.92	0.90	0.91	200
10	0.89	0.97	0.93	200
11	0.89	0.93	0.91	200
12	0.82	0.80	0.81	200
13	0.90	0.82	0.86	200
14	0.91	0.88	0.90	200
15	0.89	0.89	0.89	200
16	0.84	0.84	0.84	200
17	0.90	0.94	0.92	200
18	0.73	0.70	0.71	200
19	0.60	0.57	0.59	200
20	0.90	0.87	0.89	204
21	0.95	0.92	0.93	155
22	0.95	0.95	0.95	102
23	0.89	0.93	0.91	83
24	0.92	0.90	0.91	80
accuracy			0.82	4624
macro avg	0.83	0.83	0.83	4624
weighted avg	0.82	0.82	0.82	4624

For multinomial with predicted labels:

	precision	recall	f1-score	support
0	0.82	0.95	0.88	177
1	0.88	0.79	0.83	224
2	0.76	0.72	0.74	203
3	0.75	0.78	0.76	201
4	0.75	0.91	0.82	175
5	0.85	0.89	0.87	185
6	0.93	0.72	0.81	232
7	0.89	0.85	0.87	203
8	0.90	0.99	0.94	183
9	0.91	0.94	0.92	189
10	0.92	0.96	0.94	197
11	0.76	0.96	0.85	186
12	0.89	0.67	0.76	245
13	0.96	0.94	0.95	185
14	0.85	0.97	0.91	182
15	0.94	0.93	0.94	230
16	0.82	0.88	0.85	243
17	0.88	0.98	0.92	187
18	0.82	0.73	0.77	213
19	0.83	0.75	0.79	157
20	0.71	0.99	0.83	194
21	0.94	0.96	0.95	151
22	1.00	0.86	0.93	102
23	1.00	0.52	0.68	95
24	1.00	0.42	0.60	85
accuracy			0.85	4624
macro avg	0.87	0.84	0.84	4624
weighted avg	0.86	0.85	0.85	4624

MultinomialNB with actual labels:

	precision	recall	f1-score	support
0	0.79	0.81	0.80	200
1	0.85	0.85	0.85	200
2	0.83	0.81	0.82	200
3	0.77	0.80	0.78	200
4	0.85	0.91	0.88	200
5	0.89	0.86	0.88	200
6	0.88	0.79	0.83	200
7	0.93	0.91	0.92	200
8	0.95	0.96	0.96	200
9	0.96	0.94	0.95	200
10	0.95	0.98	0.97	200
11	0.82	0.95	0.88	200
12	0.86	0.80	0.83	200
13	0.96	0.87	0.91	200
14	0.90	0.93	0.92	200
15	0.87	0.99	0.93	200
16	0.69	0.90	0.78	200
17	0.91	0.95	0.93	200
18	0.75	0.71	0.73	200
19	0.68	0.48	0.57	200
20	0.74	1.00	0.85	204
21	0.95	0.95	0.95	155
22	1.00	0.86	0.93	102
23	1.00	0.59	0.74	83
24	1.00	0.45	0.62	80
accuracy			0.86	4624
macro avg	0.87	0.84	0.85	4624
weighted avg	0.86	0.86	0.85	4624

SGDClassifier with predicted labels:

	precision	recall	f1-score	support
0	0.81	0.92	0.86	177
1	0.89	0.76	0.82	224
2	0.80	0.81	0.80	203
3	0.79	0.74	0.76	201
4	0.80	0.91	0.85	175
5	0.81	0.90	0.85	185
6	0.89	0.83	0.86	232
7	0.88	0.90	0.89	203
8	0.89	0.97	0.93	183
9	0.91	0.95	0.93	189
10	0.95	0.96	0.95	197
11	0.87	0.95	0.91	186
12	0.83	0.67	0.74	245
13	0.88	0.98	0.93	185
14	0.88	0.97	0.92	182
15	0.98	0.90	0.94	230
16	0.90	0.77	0.83	243
17	0.88	0.98	0.93	187
18	0.78	0.62	0.69	213
19	0.66	0.76	0.71	157
20	0.91	0.98	0.95	194
21	0.93	0.97	0.95	151
22	0.99	0.99	0.99	102
23	1.00	0.85	0.92	95
24	0.96	0.91	0.93	85
accuracy			0.87	4624
macro avg	0.88	0.88	0.87	4624
weighted avg	0.87	0.87	0.87	4624

SGDClassifier with actual labels:

	precision	recall	f1-score	support
0	0.80	0.81	0.80	200
1	0.86	0.81	0.84	200
2	0.78	0.81	0.79	200
3	0.82	0.77	0.80	200
4	0.88	0.89	0.88	200
5	0.87	0.88	0.87	200
6	0.84	0.91	0.87	200
7	0.92	0.95	0.94	200
8	0.96	0.95	0.96	200
9	0.97	0.97	0.97	200
10	0.98	0.98	0.98	200
11	0.94	0.95	0.95	200
12	0.90	0.88	0.89	200
13	0.91	0.94	0.92	200
14	0.95	0.95	0.95	200
15	0.95	1.00	0.98	200
16	0.85	0.90	0.87	200
17	0.92	0.95	0.93	200
18	0.83	0.72	0.77	200
19	0.64	0.58	0.61	200
20	0.97	1.00	0.98	204
21	0.99	1.00	1.00	155
22	0.99	0.99	0.99	102
23	1.00	0.98	0.99	83
24	0.97	0.97	0.97	80
accuracy			0.89	4624
macro avg	0.90	0.90	0.90	4624
weighted avg	0.89	0.89	0.89	4624

Logistic Regression with predicted labels:

	precision	recall	f1-score	support
0	0.85	0.90	0.87	177
1	0.91	0.87	0.89	224
2	0.85	0.86	0.86	203
3	0.84	0.79	0.81	201
4	0.84	0.92	0.88	175
5	0.83	0.94	0.88	185
6	0.93	0.91	0.92	232
7	0.94	0.93	0.94	203
8	0.93	0.98	0.96	183
9	0.92	0.95	0.94	189
10	0.98	0.96	0.97	197
11	0.93	0.96	0.94	186
12	0.88	0.74	0.81	245
13	0.88	0.99	0.93	185
14	0.94	0.97	0.95	182
15	0.99	0.87	0.93	230
16	0.92	0.79	0.85	243
17	0.93	0.99	0.96	187
18	0.78	0.75	0.76	213
19	0.69	0.84	0.76	157
20	0.91	0.98	0.95	194
21	0.95	0.97	0.96	151
22	0.99	0.98	0.99	102
23	0.99	0.83	0.90	95
24	0.99	0.89	0.94	85
accuracy			0.90	4624
macro avg	0.90	0.90	0.90	4624
weighted avg	0.90	0.90	0.90	4624

Logistic Regression with actual labels:

	precision	recall	f1-score	support
0	0.84	0.79	0.81	200
1	0.80	0.85	0.82	200
2	0.77	0.79	0.78	200
3	0.78	0.74	0.76	200
4	0.91	0.86	0.88	200
5	0.81	0.85	0.83	200
6	0.79	0.90	0.84	200
7	0.91	0.91	0.91	200
8	0.98	0.94	0.96	200
9	0.96	0.94	0.95	200
10	0.99	0.96	0.98	200
11	0.96	0.92	0.94	200
12	0.83	0.86	0.85	200
13	0.91	0.94	0.93	200
14	0.97	0.92	0.94	200
15	0.98	0.99	0.99	200
16	0.84	0.86	0.85	200
17	0.94	0.94	0.94	200
18	0.74	0.75	0.74	200
19	0.64	0.61	0.63	200
20	0.97	0.99	0.98	204
21	1.00	0.99	1.00	155
22	0.99	0.98	0.99	102
23	0.99	0.95	0.97	83
24	1.00	0.96	0.98	80
accuracy			0.88	4624
macro avg	0.89	0.89	0.89	4624
eighted avg	0.88	0.88	0.88	4624

When the train is 30% and test is 70%:

Accuracy of actual and predicted:

```
accuracy = accuracy_score(y_test, test_class_predictions)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.8217882963603782
```

kNN with predicted labels:

	precision	recall	f1-score	support
0	0.63	0.81	0.71	593
1	0.63	0.69	0.66	835
2	0.62	0.69	0.65	743
3	0.64	0.72	0.68	760
4	0.65	0.68	0.67	647
5	0.69	0.76	0.72	555
6	0.72	0.48	0.57	741
7	0.80	0.80	0.80	696
8	0.81	0.89	0.85	662
9	0.83	0.90	0.87	661
10	0.84	0.93	0.88	704
11	0.84	0.91	0.87	658
12	0.79	0.50	0.61	853
13	0.87	0.83	0.85	629
14	0.84	0.91	0.87	661
15	0.90	0.77	0.83	833
16	0.83	0.70	0.76	835
17	0.83	0.95	0.88	629
18	0.68	0.59	0.63	747
19	0.59	0.66	0.62	560
20	0.88	0.89	0.88	673
21	0.87	0.93	0.90	521
22	0.96	0.93	0.94	361
23	0.92	0.83	0.87	340
24	0.88	0.73	0.80	286
accuracy			0.77	16183
macro avg	0.78	0.78	0.78	16183
eighted avg	0.77	0.77	0.77	16183

kNN with actual labels:

	precision	recall	f1-score	support
0	0.67	0.72	0.69	700
1	0.57	0.74	0.64	700
2	0.59	0.70	0.64	700
3	0.54	0.66	0.59	700
4	0.66	0.64	0.65	700
5	0.75	0.66	0.70	700
6	0.70	0.49	0.57	700
7	0.80	0.80	0.80	700
8	0.83	0.87	0.85	700
9	0.85	0.87	0.86	700
10	0.85	0.94	0.89	700
11	0.87	0.89	0.88	700
12	0.79	0.61	0.69	700
13	0.90	0.78	0.84	700
14	0.85	0.87	0.86	700
15	0.83	0.86	0.85	698
16	0.77	0.77	0.77	700
17	0.88	0.91	0.89	700
18	0.69	0.64	0.66	700
19	0.59	0.53	0.56	700
20	0.90	0.86	0.88	714
21	0.89	0.91	0.90	540
22	0.95	0.93	0.94	358
23	0.85	0.89	0.87	292
24	0.87	0.73	0.80	281
accuracy			0.76	16183
macro avg	0.78	0.77	0.77	16183
eighted avg	0.77	0.76	0.76	16183

MultinomialNB with predicted labels:

	precision	recall	f1-score	support
0	0.82	0.95	0.88	593
1	0.91	0.77	0.84	835
2	0.84	0.77	0.80	743
3	0.80	0.89	0.85	760
4	0.84	0.92	0.88	647
5	0.81	0.93	0.87	555
6	0.94	0.78	0.86	741
7	0.88	0.87	0.87	696
8	0.92	0.97	0.94	662
9	0.90	0.97	0.93	661
10	0.90	0.97	0.94	704
11	0.72	0.97	0.83	658
12	0.94	0.62	0.75	853
13	0.95	0.93	0.94	629
14	0.89	0.97	0.93	661
15	0.91	0.95	0.93	833
16	0.81	0.95	0.88	835
17	0.85	0.99	0.91	629
18	0.87	0.70	0.78	747
19	0.90	0.75	0.82	560
20	0.65	1.00	0.79	673
21	0.93	0.98	0.95	521
22	1.00	0.71	0.83	361
23	1.00	0.38	0.55	340
24	1.00	0.13	0.23	286
accuracy			0.86	16183
macro avg	0.88	0.83	0.83	16183
weighted avg	0.87	0.86	0.85	16183

Multinomial with actual labels:

	precision	recall	f1-score	support
0	0.75	0.74	0.75	700
1	0.76	0.77	0.77	700
2	0.77	0.75	0.76	700
3	0.66	0.80	0.72	700
4	0.84	0.86	0.85	700
5	0.88	0.80	0.84	700
6	0.87	0.77	0.82	700
7	0.90	0.88	0.89	700
8	0.94	0.94	0.94	700
9	0.94	0.96	0.95	700
10	0.89	0.97	0.93	700
11	0.75	0.95	0.84	700
12	0.89	0.72	0.80	700
13	0.97	0.86	0.91	700
14	0.91	0.93	0.92	700
15	0.80	0.99	0.89	698
16	0.65	0.92	0.76	700
17	0.89	0.93	0.91	700
18	0.76	0.65	0.70	700
19	0.63	0.43	0.51	700
20	0.69	0.99	0.81	714
21	0.94	0.96	0.95	540
22	1.00	0.71	0.83	358
23	1.00	0.44	0.61	292
24	1.00	0.13	0.23	281
accuracy			0.82	16183
macro avg	0.84	0.79	0.80	16183
weighted avg	0.83	0.82	0.81	16183

SGDClassifier with predicted labels:

	precision	recall	f1-score	support
0	0.80	0.91	0.85	593
1	0.83	0.76	0.79	835
2	0.84	0.80	0.82	743
3	0.84	0.77	0.80	760
4	0.82	0.90	0.86	647
5	0.77	0.95	0.85	555
6	0.87	0.86	0.86	741
7	0.90	0.90	0.90	696
8	0.92	0.97	0.94	662
9	0.92	0.97	0.94	661
10	0.94	0.96	0.95	704
11	0.90	0.95	0.93	658
12	0.89	0.67	0.77	853
13	0.86	0.97	0.91	629
14	0.88	0.97	0.92	661
15	0.97	0.87	0.92	833
16	0.92	0.81	0.86	835
17	0.89	0.98	0.93	629
18	0.80	0.71	0.76	747
19	0.70	0.76	0.73	560
20	0.92	0.99	0.96	673
21	0.92	0.97	0.95	521
22	0.98	0.99	0.99	361
23	0.99	0.85	0.91	340
24	0.96	0.83	0.89	286
accuracy			0.88	16183
macro avg	0.88	0.88	0.88	16183
ıeighted avg	0.88	0.88	0.87	16183

SGDClassifier with actual labels:

	precision	recall	f1-score	support
0	0.81	0.76	0.78	700
1	0.77	0.81	0.79	700
2	0.79	0.78	0.78	700
3	0.74	0.76	0.75	700
4	0.86	0.87	0.86	700
5	0.84	0.82	0.83	700
6	0.80	0.85	0.83	700
7	0.90	0.89	0.90	700
8	0.95	0.95	0.95	700
9	0.94	0.95	0.95	700
10	0.95	0.97	0.96	700
11	0.95	0.94	0.94	700
12	0.87	0.80	0.83	700
13	0.91	0.94	0.92	700
14	0.92	0.95	0.93	700
15	0.94	0.99	0.97	698
16	0.81	0.85	0.83	700
17	0.93	0.92	0.92	700
18	0.74	0.70	0.72	700
19	0.66	0.60	0.63	700
20	0.95	0.96	0.96	714
21	0.96	0.97	0.97	540
22	0.98	1.00	0.99	358
23	0.93	0.94	0.93	292
24	0.98	0.87	0.92	281
accuracy			0.87	16183
macro avg	0.88	0.87	0.87	16183
ıeighted avg	0.87	0.87	0.87	16183

LogisticRegression with predicted labels:

	precision	recall	f1-score	support
0	0.86	0.94	0.90	593
1	0.92	0.89	0.90	835
2	0.93	0.88	0.90	743
3	0.91	0.88	0.89	760
4	0.90	0.95	0.92	647
5	0.83	0.99	0.90	555
6	0.91	0.95	0.93	741
7	0.95	0.96	0.96	696
8	0.96	0.98	0.97	662
9	0.95	0.98	0.96	661
10	0.96	0.98	0.97	704
11	0.95	0.95	0.95	658
12	0.95	0.80	0.87	853
13	0.89	0.99	0.94	629
14	0.94	0.98	0.96	661
15	0.99	0.91	0.95	833
16	0.97	0.87	0.92	835
17	0.93	0.99	0.96	629
18	0.87	0.85	0.86	747
19	0.82	0.84	0.83	560
20	0.89	1.00	0.94	673
21	0.93	0.98	0.96	521
22	1.00	0.97	0.98	361
23	1.00	0.75	0.86	340
24	1.00	0.74	0.85	286
accuracy			0.92	16183
macro avg	0.93	0.92	0.92	16183
weighted avg	0.93	0.92	0.92	16183

LogisticRegression with actual labels:

	precision	recall	f1-score	support
0	0.80	0.74	0.77	700
1	0.70	0.81	0.75	700
2	0.76	0.76	0.76	700
3	0.72	0.75	0.73	700
4	0.85	0.83	0.84	700
5	0.82	0.78	0.80	700
6	0.77	0.85	0.80	700
7	0.88	0.88	0.88	700
8	0.95	0.93	0.94	700
9	0.95	0.93	0.94	700
10	0.95	0.97	0.96	700
11	0.96	0.90	0.93	700
12	0.78	0.80	0.79	700
13	0.92	0.92	0.92	700
14	0.93	0.92	0.93	700
15	0.91	0.99	0.95	698
16	0.78	0.84	0.81	700
17	0.94	0.91	0.92	700
18	0.71	0.73	0.72	700
19	0.66	0.54	0.60	700
20	0.92	0.98	0.95	714
21	0.96	0.97	0.97	540
22	0.99	0.97	0.98	358
23	0.98	0.86	0.91	292
24	0.99	0.75	0.85	281
accuracy			0.85	16183
macro avg	0.86	0.85	0.86	16183
weighted avg	0.85	0.85	0.85	16183

TFIDF with n-grams:

Train – 80% and test – 20%

Accuracy of predicted and actual labels:

```
accuracy = accuracy_score(y_test, test_class_predictions)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.8161764705882353
```

kNN classifier with predicted labels:

	precision	recall	f1-score	support
0	0.62	0.84	0.71	168
1	0.56	0.59	0.58	227
2	0.54	0.61	0.57	207
3	0.61	0.68	0.64	202
4	0.60	0.71	0.65	174
5	0.69	0.73	0.71	171
6	0.65	0.49	0.56	245
7	0.75	0.74	0.75	198
8	0.76	0.83	0.80	183
9	0.79	0.84	0.81	190
10	0.87	0.90	0.89	197
11	0.84	0.89	0.87	186
12	0.74	0.47	0.57	273
13	0.87	0.79	0.83	185
14	0.84	0.89	0.86	178
15	0.93	0.84	0.88	223
16	0.79	0.70	0.74	230
17	0.82	0.96	0.88	183
18	0.61	0.55	0.58	211
19	0.62	0.64	0.63	168
20	0.89	0.92	0.91	199
21	0.88	0.92	0.90	145
22	0.97	0.92	0.94	102
23	0.94	0.87	0.90	92
24	0.92	0.82	0.87	87
accuracy			0.75	4624
macro avg	0.76	0.77	0.76	4624
eighted avg	0.75	0.75	0.74	4624

kNN classifier with actual labels:

	precision	recall	f1-score	support
0	0.66	0.75	0.70	200
1	0.55	0.67	0.60	200
2	0.60	0.70	0.65	200
3	0.61	0.69	0.65	200
4	0.64	0.66	0.65	200
5	0.83	0.74	0.78	200
6	0.60	0.56	0.58	200
7	0.79	0.77	0.78	200
8	0.82	0.81	0.82	200
9	0.84	0.84	0.84	200
10	0.89	0.91	0.90	200
11	0.91	0.90	0.91	200
12	0.82	0.70	0.76	200
13	0.87	0.73	0.79	200
14	0.90	0.85	0.87	200
15	0.91	0.91	0.91	200
16	0.80	0.81	0.81	200
17	0.86	0.92	0.89	200
18	0.70	0.67	0.69	200
19	0.61	0.53	0.56	200
20	0.90	0.91	0.90	204
21	0.95	0.94	0.94	155
22	0.98	0.93	0.95	102
23	0.92	0.94	0.93	83
24	0.88	0.85	0.87	80
accuracy			0.78	4624
macro avg	0.79	0.79	0.79	4624
ighted avg	0.78	0.78	0.78	4624

MultinomialNB with predicted labels:

	precision	recall	f1-score	support
0	0.83	0.96	0.89	168
1	0.85	0.79	0.82	227
2	0.79	0.75	0.77	207
3	0.72	0.83	0.77	202
4	0.78	0.88	0.83	174
5	0.78	0.93	0.85	171
6	0.87	0.83	0.85	245
7	0.84	0.86	0.85	198
8	0.85	0.97	0.91	183
9	0.93	0.93	0.93	190
10	0.95	0.97	0.96	197
11	0.86	0.96	0.91	186
12	0.92	0.61	0.74	273
13	0.98	0.96	0.97	185
14	0.93	0.95	0.94	178
15	0.93	0.96	0.95	223
16	0.90	0.89	0.89	230
17	0.91	0.99	0.95	183
18	0.88	0.72	0.79	211
19	0.84	0.74	0.78	168
20	0.83	0.99	0.90	199
21	0.94	0.99	0.97	145
22	1.00	0.98	0.99	102
23	1.00	0.74	0.85	92
24	1.00	0.78	0.88	87
accuracy			0.87	4624
macro avg	0.89	0.88	0.88	4624
weighted avg	0.88	0.87	0.87	4624

MultinomialNB with actual labels:

	precision	recall	f1-score	support
0	0.78	0.77	0.77	200
1	0.74	0.78	0.76	200
2	0.77	0.75	0.76	200
3	0.70	0.81	0.75	200
4	0.83	0.81	0.82	200
5	0.83	0.85	0.84	200
6	0.74	0.86	0.80	200
7	0.88	0.89	0.88	200
8	0.89	0.93	0.91	200
9	0.92	0.88	0.90	200
10	0.94	0.95	0.95	200
11	0.91	0.95	0.93	200
12	0.82	0.74	0.78	200
13	0.95	0.86	0.91	200
14	0.96	0.87	0.91	200
15	0.87	0.99	0.93	200
16	0.73	0.82	0.77	200
17	0.93	0.93	0.93	200
18	0.75	0.65	0.69	200
19	0.66	0.48	0.56	200
20	0.85	1.00	0.92	204
21	0.97	0.96	0.97	155
22	1.00	0.98	0.99	102
23	1.00	0.82	0.90	83
24	1.00	0.85	0.92	80
accuracy			0.84	4624
macro avg	0.86	0.85	0.85	4624
weighted avg	0.85	0.84	0.84	4624

SGDClassifier with predicted labels:

	precision	recall	f1-score	support
0	0.80	0.93	0.86	168
1	0.86	0.72	0.79	227
2	0.79	0.83	0.81	207
3	0.80	0.77	0.79	202
4	0.80	0.90	0.85	174
5	0.80	0.89	0.84	171
6	0.88	0.81	0.84	245
7	0.85	0.88	0.87	198
8	0.88	0.95	0.91	183
9	0.89	0.93	0.91	190
10	0.94	0.96	0.95	197
11	0.85	0.97	0.91	186
12	0.89	0.58	0.71	273
13	0.85	0.95	0.90	185
14	0.87	0.98	0.92	178
15	0.99	0.92	0.95	223
16	0.89	0.80	0.84	230
17	0.86	0.98	0.92	183
18	0.79	0.64	0.71	211
19	0.68	0.81	0.74	168
20	0.94	0.99	0.96	199
21	0.91	0.99	0.95	145
22	0.99	0.99	0.99	102
23	0.98	0.88	0.93	92
24	0.97	0.89	0.93	87
accuracy			0.86	4624
macro avg	0.87	0.88	0.87	4624
weighted avg	0.87	0.86	0.86	4624

SGDClassifier with actual labels:

	precision	recall	f1-score	support
0	0.82	0.80	0.81	200
1	0.81	0.78	0.80	200
2	0.77	0.80	0.79	200
3	0.79	0.77	0.78	200
4	0.86	0.83	0.85	200
5	0.88	0.83	0.86	200
6	0.77	0.90	0.83	200
7	0.89	0.92	0.90	200
8	0.96	0.94	0.95	200
9	0.96	0.94	0.95	200
10	0.96	0.97	0.97	200
11	0.93	0.97	0.95	200
12	0.87	0.80	0.83	200
13	0.91	0.94	0.92	200
14	0.94	0.94	0.94	200
15	0.96	1.00	0.98	200
16	0.81	0.85	0.83	200
17	0.91	0.95	0.93	200
18	0.80	0.68	0.73	200
19	0.65	0.65	0.65	200
20	0.96	0.99	0.98	204
21	0.97	0.99	0.98	155
22	0.99	0.99	0.99	102
23	0.98	0.98	0.98	83
24	0.99	0.97	0.98	80
accuracy			0.88	4624
macro avg	0.89	0.89	0.89	4624
weighted avg	0.88	0.88	0.88	4624

LogisticRegression with predicted labels:

	precision	recall	f1-score	support
0	0.82	0.95	0.88	168
1	0.89	0.85	0.87	227
2	0.87	0.86	0.87	207
3	0.80	0.81	0.80	202
4	0.82	0.90	0.86	174
5	0.85	0.94	0.89	171
6	0.92	0.86	0.89	245
7	0.91	0.93	0.92	198
8	0.92	0.96	0.94	183
9	0.91	0.95	0.93	190
10	0.97	0.96	0.96	197
11	0.91	0.96	0.93	186
12	0.91	0.70	0.79	273
13	0.91	0.96	0.93	185
14	0.91	0.96	0.93	178
15	0.99	0.91	0.95	223
16	0.91	0.79	0.85	230
17	0.91	0.99	0.95	183
18	0.81	0.76	0.78	211
19	0.70	0.82	0.76	168
20	0.94	0.98	0.96	199
21	0.93	0.99	0.96	145
22	0.99	0.99	0.99	102
23	0.99	0.87	0.92	92
24	0.99	0.90	0.94	87
accuracy			0.89	4624
macro avg	0.90	0.90	0.90	4624
eighted avg	0.90	0.89	0.89	4624

LogisticRegression with actual labels:

	precision	recall	f1-score	support
0	0.82	0.80	0.81	200
1	0.76	0.82	0.79	200
2	0.75	0.77	0.76	200
3	0.75	0.77	0.76	200
4	0.88	0.83	0.86	200
5	0.88	0.83	0.85	200
6	0.77	0.88	0.82	200
7	0.88	0.90	0.89	200
8	0.96	0.93	0.94	200
9	0.91	0.90	0.90	200
10	0.98	0.95	0.97	200
11	0.96	0.94	0.95	200
12	0.78	0.83	0.81	200
13	0.93	0.90	0.91	200
14	0.97	0.91	0.94	200
15	0.97	0.99	0.98	200
16	0.83	0.82	0.83	200
17	0.94	0.94	0.94	200
18	0.74	0.73	0.73	200
19	0.66	0.65	0.65	200
20	0.98	0.99	0.98	204
21	1.00	0.99	0.99	155
22	0.99	0.99	0.99	102
23	0.99	0.96	0.98	83
24	1.00	0.99	0.99	80
accuracy			0.87	4624
macro avg	0.88	0.88	0.88	4624
eighted avg	0.87	0.87	0.87	4624

With train – 30% and test 70%

Accuracy between predicted and actual labels:

kNN with predicted labels:

	precision	recall	f1-score	support
0	0.59	0.78	0.67	590
1	0.50	0.62	0.55	829
2	0.54	0.65	0.59	717
3	0.57	0.69	0.62	737
4	0.54	0.61	0.57	636
5	0.62	0.72	0.66	547
6	0.57	0.49	0.53	760
7	0.74	0.72	0.73	672
8	0.74	0.76	0.75	685
9	0.72	0.82	0.77	647
10	0.81	0.89	0.85	691
11	0.82	0.89	0.85	637
12	0.76	0.36	0.49	1079
13	0.81	0.76	0.78	596
14	0.81	0.85	0.83	632
15	0.91	0.83	0.87	802
16	0.81	0.68	0.74	805
17	0.79	0.94	0.86	603
18	0.72	0.53	0.61	742
19	0.61	0.59	0.60	586
20	0.89	0.89	0.89	688
21	0.90	0.91	0.90	520
22	0.97	0.93	0.95	363
23	0.93	0.85	0.89	324
24	0.92	0.74	0.82	295
accuracy			0.72	16183
macro avg	0.74	0.74	0.73	16183
weighted avg	0.73	0.72	0.72	16183

kNN with actual labels:

	precision	recall	f1-score	support
0	0.62	0.70	0.66	700
1	0.46	0.67	0.55	700
2	0.52	0.64	0.58	700
3	0.49	0.62	0.55	700
4	0.54	0.56	0.55	700
5	0.69	0.62	0.65	700
6	0.53	0.49	0.51	700
7	0.77	0.71	0.74	700
8	0.76	0.77	0.76	700
9	0.73	0.77	0.75	700
10	0.80	0.88	0.84	700
11	0.86	0.84	0.85	700
12	0.69	0.50	0.58	700
13	0.87	0.69	0.77	700
14	0.82	0.78	0.80	700
15	0.86	0.91	0.88	698
16	0.76	0.73	0.75	700
17	0.85	0.87	0.86	700
18	0.75	0.59	0.66	700
19	0.60	0.48	0.53	700
20	0.90	0.87	0.89	714
21	0.92	0.90	0.91	540
22	0.96	0.94	0.95	358
23	0.88	0.89	0.89	292
24	0.89	0.75	0.81	281
accuracy			0.72	16183
macro avg	0.74	0.73	0.73	16183
weighted avg	0.73	0.72	0.72	16183

MultinomialNB with predicted labels:

	precision	recall	f1-score	support
0	0.83	0.94	0.88	590
1	0.87	0.78	0.82	829
2	0.80	0.81	0.81	717
3	0.76	0.92	0.83	737
4	0.84	0.86	0.85	636
5	0.78	0.95	0.86	547
6	0.86	0.90	0.88	760
7	0.86	0.86	0.86	672
8	0.85	0.94	0.89	685
9	0.90	0.97	0.93	647
10	0.94	0.97	0.96	691
11	0.84	0.97	0.90	637
12	0.95	0.55	0.70	1079
13	0.93	0.92	0.92	596
14	0.91	0.95	0.93	632
15	0.91	0.97	0.94	802
16	0.87	0.94	0.90	805
17	0.88	0.99	0.93	603
18	0.94	0.70	0.80	742
19	0.91	0.75	0.82	586
20	0.75	1.00	0.86	688
21	0.89	0.98	0.94	520
22	1.00	0.94	0.97	363
23	1.00	0.58	0.73	324
24	1.00	0.46	0.63	295
accuracy			0.87	16183
macro avg	0.88	0.86	0.86	16183
weighted avg	0.88	0.87	0.86	16183

MultinomialNB with actual labels:

	precision	recall	f1-score	support
0	0.78	0.74	0.76	700
1	0.70	0.74	0.72	700
2	0.72	0.75	0.74	700
3	0.63	0.80	0.70	700
4	0.83	0.77	0.80	700
5	0.83	0.79	0.81	700
6	0.77	0.87	0.81	700
7	0.87	0.83	0.85	700
8	0.84	0.91	0.88	700
9	0.91	0.91	0.91	700
10	0.93	0.95	0.94	700
11	0.88	0.93	0.90	700
12	0.78	0.70	0.74	700
13	0.94	0.79	0.86	700
14	0.93	0.88	0.90	700
15	0.81	1.00	0.89	698
16	0.70	0.87	0.77	700
17	0.92	0.89	0.90	700
18	0.81	0.64	0.71	700
19	0.64	0.44	0.52	700
20	0.78	1.00	0.88	714
21	0.92	0.98	0.95	540
22	0.99	0.95	0.97	358
23	1.00	0.64	0.78	292
24	1.00	0.48	0.65	281
accuracy			0.82	16183
macro avg	0.84	0.81	0.81	16183
weighted avg	0.82	0.82	0.81	16183

SGDClassifier with predicted labels:

	precision	recall	f1-score	support
0	0.78	0.88	0.83	590
1	0.81	0.71	0.75	829
2	0.84	0.80	0.82	717
3	0.79	0.76	0.78	737
4	0.78	0.86	0.82	636
5	0.76	0.93	0.84	547
6	0.84	0.84	0.84	760
7	0.85	0.89	0.87	672
8	0.89	0.91	0.90	685
9	0.89	0.96	0.92	647
10	0.93	0.97	0.95	691
11	0.87	0.97	0.91	637
12	0.89	0.52	0.66	1079
13	0.78	0.94	0.85	596
14	0.84	0.96	0.90	632
15	0.99	0.89	0.94	802
16	0.89	0.81	0.85	805
17	0.83	0.99	0.91	603
18	0.79	0.68	0.74	742
19	0.69	0.77	0.73	586
20	0.94	0.98	0.96	688
21	0.90	0.98	0.94	520
22	0.99	0.99	0.99	363
23	0.98	0.89	0.93	324
24	0.95	0.82	0.88	295
accuracy			0.85	16183
macro avg	0.86	0.87	0.86	16183
weighted avg	0.86	0.85	0.85	16183

SGDClassifier with actual labels:

	precision	recall	f1-score	support
0	0.80	0.76	0.78	700
1	0.74	0.77	0.75	700
2	0.79	0.75	0.77	700
3	0.73	0.72	0.73	700
4	0.82	0.84	0.83	700
5	0.81	0.80	0.81	700
6	0.80	0.86	0.83	700
7	0.86	0.88	0.87	700
8	0.94	0.93	0.93	700
9	0.92	0.94	0.93	700
10	0.94	0.96	0.95	700
11	0.92	0.94	0.93	700
12	0.83	0.75	0.79	700
13	0.89	0.90	0.90	700
14	0.89	0.93	0.91	700
15	0.96	0.99	0.98	698
16	0.81	0.84	0.83	700
17	0.91	0.93	0.92	700
18	0.76	0.68	0.72	700
19	0.66	0.61	0.63	700
20	0.96	0.96	0.96	714
21	0.96	0.99	0.97	540
22	0.98	0.99	0.99	358
23	0.92	0.95	0.93	292
24	0.96	0.88	0.92	281
accuracy			0.86	16183
macro avg	0.86	0.86	0.86	16183
weighted avg	0.86	0.86	0.86	16183

LogisticRegression with predicted labels:

	precision	recall	f1-score	support
0	0.84	0.94	0.89	590
1	0.90	0.84	0.87	829
2	0.90	0.88	0.89	717
3	0.86	0.88	0.87	737
4	0.86	0.94	0.90	636
5	0.82	0.98	0.89	547
6	0.92	0.92	0.92	760
7	0.91	0.95	0.93	672
8	0.94	0.94	0.94	685
9	0.92	0.98	0.95	647
10	0.96	0.99	0.97	691
11	0.94	0.98	0.96	637
12	0.96	0.65	0.78	1079
13	0.85	0.97	0.90	596
14	0.91	0.98	0.94	632
15	0.99	0.93	0.96	802
16	0.95	0.86	0.90	805
17	0.89	1.00	0.94	603
18	0.87	0.84	0.85	742
19	0.81	0.86	0.83	586
20	0.92	0.99	0.96	688
21	0.93	0.98	0.96	520
22	1.00	0.98	0.99	363
23	1.00	0.83	0.91	324
24	1.00	0.78	0.88	295
accuracy			0.91	16183
macro avg	0.91	0.91	0.91	16183
weighted avg	0.91	0.91	0.91	16183

LogisticRegression with actual labels:

	precision	recall	f1-score	support
0	0.79	0.75	0.77	700
1	0.70	0.78	0.74	700
2	0.75	0.76	0.75	700
3	0.70	0.75	0.73	700
4	0.82	0.81	0.81	700
5	0.83	0.77	0.80	700
6	0.78	0.85	0.82	700
7	0.86	0.86	0.86	700
8	0.92	0.91	0.92	700
9	0.93	0.92	0.92	700
10	0.94	0.96	0.95	700
11	0.95	0.91	0.93	700
12	0.73	0.77	0.75	700
13	0.90	0.88	0.89	700
14	0.93	0.90	0.91	700
15	0.93	0.99	0.96	698
16	0.79	0.82	0.81	700
17	0.94	0.90	0.92	700
18	0.71	0.73	0.72	700
19	0.66	0.59	0.62	700
20	0.94	0.97	0.96	714
21	0.96	0.98	0.97	540
22	0.99	0.98	0.99	358
23	0.97	0.89	0.93	292
24	0.99	0.81	0.89	281
accuracy			0.85	16183
macro avg	0.86	0.85	0.85	16183
weighted avg	0.85	0.85	0.85	16183

With unlabeled dataset and labeled dataset:

kNN

	precision	recall	f1-score	support
0	0.59	0.71	0.64	267
1	0.58	0.75	0.66	264
2	0.64	0.77	0.70	227
3	0.64	0.74	0.69	233
4	0.73	0.70	0.72	229
5	0.64	0.64	0.64	307
6	0.61	0.55	0.58	248
7	0.73	0.76	0.75	281
8	0.78	0.75	0.76	323
9	0.79	0.75	0.77	300
10	0.72	0.81	0.76	285
11	0.74	0.80	0.77	280
12	0.69	0.68	0.69	293
13	0.71	0.68	0.69	371
14	0.79	0.74	0.76	312
15	0.82	0.80	0.81	238
16	0.75	0.69	0.72	337
17	0.73	0.76	0.74	377
18	0.64	0.55	0.59	316
19	0.52	0.45	0.48	301
20	0.73	0.77	0.75	375
21	0.77	0.72	0.74	311
22	0.80	0.67	0.73	230
23	0.83	0.69	0.75	140
24	0.68	0.62	0.65	203
accuracy			0.70	7048
macro avg	0.71	0.70	0.70	7048
weighted avg	0.71	0.70	0.70	7048

MultinomialNB:

	precision	recall	f1-score	support
0	0.77	0.73	0.75	267
1	0.81	0.78	0.79	264
2	0.79	0.76	0.78	227
3	0.79	0.75	0.77	233
4	0.88	0.79	0.83	229
5	0.80	0.82	0.81	307
6	0.84	0.73	0.78	248
7	0.94	0.86	0.90	281
8	0.87	0.85	0.86	323
9	0.92	0.85	0.88	300
10	0.91	0.85	0.88	285
11	0.94	0.85	0.89	280
12	0.87	0.76	0.81	293
13	0.64	0.90	0.75	371
14	0.93	0.82	0.87	312
15	0.87	0.88	0.87	238
16	0.70	0.85	0.77	337
17	0.70	0.93	0.80	377
18	0.73	0.69	0.71	316
19	0.67	0.46	0.55	301
20	0.74	0.95	0.83	375
21	0.76	0.95	0.84	311
22	0.93	0.85	0.89	230
23	1.00	0.59	0.74	140
24	0.95	0.61	0.74	203
accuracy			0.81	7048
macro avg	0.83	0.79	0.80	7048
weighted avg	0.82	0.81	0.80	7048

SGDClassifier:

	precision	recall	f1-score	support
0	0.84	0.80	0.82	267
1	0.85	0.84	0.84	264
2	0.82	0.82	0.82	227
3	0.84	0.77	0.81	233
4	0.87	0.84	0.86	229
5	0.85	0.85	0.85	307
6	0.84	0.87	0.86	248
7	0.92	0.90	0.91	281
8	0.89	0.90	0.90	323
9	0.92	0.93	0.93	300
10	0.91	0.95	0.93	285
11	0.94	0.95	0.94	280
12	0.87	0.87	0.87	293
13	0.96	0.84	0.90	371
14	0.91	0.90	0.91	312
15	0.90	0.97	0.93	238
16	0.86	0.87	0.86	337
17	0.89	0.96	0.92	377
18	0.84	0.72	0.77	316
19	0.65	0.67	0.66	301
20	0.88	0.95	0.92	375
21	0.93	0.95	0.94	311
22	0.84	0.94	0.89	230
23	0.95	0.91	0.93	140
24	0.97	0.86	0.91	203
accuracy			0.88	7048
macro avg	0.88	0.87	0.87	7048
weighted avg	0.88	0.88	0.87	7048

LogisticRegression:

	precision	recall	f1-score	support
0	0.82	0.77	0.80	267
1	0.81	0.85	0.83	264
2	0.79	0.79	0.79	227
3	0.80	0.76	0.78	233
4	0.88	0.79	0.83	229
5	0.85	0.83	0.84	307
6	0.83	0.82	0.82	248
7	0.93	0.90	0.92	281
8	0.89	0.89	0.89	323
9	0.94	0.90	0.92	300
10	0.95	0.92	0.93	285
11	0.97	0.90	0.93	280
12	0.85	0.83	0.84	293
13	0.76	0.92	0.83	371
14	0.92	0.87	0.89	312
15	0.94	0.91	0.93	238
16	0.86	0.86	0.86	337
17	0.83	0.92	0.88	377
18	0.75	0.76	0.75	316
19	0.69	0.65	0.67	301
20	0.85	0.94	0.89	375
21	0.85	0.96	0.90	311
22	0.92	0.86	0.89	230
23	1.00	0.78	0.88	140
24	0.93	0.87	0.90	203
accuracy			0.86	7048
macro avg	0.86	0.85	0.86	7048
weighted avg	0.86	0.86	0.86	7048

Unlabeled with tfidf n-grams
kNN

	precision	recall	f1-score	support
0	0.57	0.69	0.62	300
1	0.52	0.71	0.60	256
2	0.64	0.73	0.68	223
3	0.54	0.66	0.59	270
4	0.56	0.56	0.56	361
5	0.62	0.64	0.63	333
6	0.55	0.57	0.56	273
7	0.73	0.77	0.75	240
8	0.74	0.71	0.72	268
9	0.69	0.65	0.67	268
10	0.74	0.76	0.75	288
11	0.68	0.67	0.67	340
12	0.69	0.61	0.65	304
13	0.71	0.67	0.69	341
14	0.70	0.65	0.67	375
15	0.91	0.85	0.88	243
16	0.74	0.77	0.75	302
17	0.77	0.75	0.76	330
18	0.66	0.63	0.64	373
19	0.57	0.44	0.50	281
20	0.78	0.74	0.76	337
21	0.77	0.76	0.76	243
22	0.78	0.72	0.75	170
23	0.89	0.71	0.79	122
24	0.67	0.59	0.63	207
accuracy			0.68	7048
macro avg	0.69	0.68	0.68	7048
weighted avg	0.68	0.68	0.68	7048

MultinomialNB:

	precision	recall	f1-score	support
0	0.76	0.77	0.77	300
1	0.77	0.75	0.76	256
2	0.85	0.73	0.79	223
3	0.71	0.70	0.71	270
4	0.49	0.78	0.60	361
5	0.77	0.79	0.78	333
6	0.76	0.77	0.76	273
7	0.91	0.83	0.87	240
8	0.89	0.78	0.83	268
9	0.90	0.77	0.83	268
10	0.87	0.88	0.87	288
11	0.83	0.84	0.83	340
12	0.87	0.69	0.77	304
13	0.90	0.82	0.86	341
14	0.63	0.84	0.72	375
15	0.86	0.87	0.87	243
16	0.80	0.82	0.81	302
17	0.88	0.81	0.84	330
18	0.66	0.73	0.69	373
19	0.72	0.44	0.55	281
20	0.75	0.88	0.81	337
21	0.95	0.88	0.92	243
22	0.96	0.76	0.85	170
23	1.00	0.66	0.79	122
24	0.84	0.77	0.80	207
accuracy			0.78	7048
macro avg	0.81	0.77	0.79	7048
weighted avg	0.80	0.78	0.78	7048

SGDClassifier:

	precision	recall	f1-score	support
0	0.85	0.79	0.82	300
1	0.82	0.79	0.80	256
2	0.87	0.79	0.83	223
3	0.75	0.80	0.78	270
4	0.85	0.82	0.83	361
5	0.89	0.86	0.87	333
6	0.83	0.88	0.85	273
7	0.85	0.90	0.88	240
8	0.90	0.90	0.90	268
9	0.95	0.94	0.95	268
10	0.94	0.94	0.94	288
11	0.93	0.91	0.92	340
12	0.86	0.81	0.84	304
13	0.88	0.92	0.90	341
14	0.81	0.91	0.86	375
15	0.94	0.94	0.94	243
16	0.85	0.90	0.87	302
17	0.91	0.90	0.90	330
18	0.85	0.76	0.80	373
19	0.71	0.66	0.68	281
20	0.86	0.95	0.91	337
21	0.95	0.96	0.96	243
22	0.94	0.95	0.95	170
23	0.98	0.85	0.91	122
24	0.84	0.91	0.87	207
accuracy			0.87	7048
macro avg	0.87	0.87	0.87	7048
weighted avg	0.87	0.87	0.87	7048

LogisticRegression:

	precision	recall	f1-score	support
0	0.82	0.81	0.82	300
1	0.82	0.83	0.82	256
2	0.83	0.78	0.81	223
3	0.78	0.77	0.78	270
4	0.80	0.81	0.80	361
5	0.86	0.86	0.86	333
6	0.79	0.84	0.81	273
7	0.90	0.88	0.89	240
8	0.96	0.83	0.89	268
9	0.93	0.87	0.90	268
10	0.94	0.92	0.93	288
11	0.92	0.89	0.91	340
12	0.79	0.83	0.81	304
13	0.84	0.92	0.88	341
14	0.81	0.91	0.85	375
15	0.99	0.90	0.94	243
16	0.84	0.86	0.85	302
17	0.90	0.90	0.90	330
18	0.75	0.79	0.77	373
19	0.71	0.65	0.68	281
20	0.83	0.94	0.88	337
21	0.98	0.94	0.96	243
22	0.97	0.86	0.92	170
23	0.99	0.77	0.87	122
24	0.84	0.87	0.86	207
accuracy			0.85	7048
macro avg	0.86	0.85	0.86	7048
weighted avg	0.86	0.85	0.85	7048

Using CountVectorizer:

Train – 80% and test – 20%
Accuracy between actual and predicted labels:

```
accuracy = accuracy_score(y_test, test_class_predictions)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.6535467128027682
```

kNN with predicted labels:

	precision	recall	f1-score	support
0	0.31	0.62	0.42	173
1	0.11	0.54	0.18	81
2	0.02	0.58	0.04	12
3	0.41	0.30	0.35	194
4	0.22	0.51	0.30	185
5	0.18	0.40	0.25	213
6	0.49	0.29	0.36	512
7	0.55	0.50	0.52	155
8	0.62	0.24	0.35	370
9	0.79	0.54	0.64	171
10	0.67	0.69	0.68	145
11	0.77	0.60	0.68	139
12	0.60	0.15	0.24	515
13	0.67	0.45	0.54	151
14	0.57	0.69	0.62	124
15	0.87	0.80	0.84	220
16	0.62	0.55	0.58	188
17	0.59	0.84	0.69	135
18	0.62	0.35	0.45	199
19	0.38	0.44	0.41	120
20	0.87	0.43	0.58	199
21	0.90	0.43	0.58	140
22	1.00	0.33	0.49	101
23	0.98	0.42	0.58	101
24	1.00	0.30	0.46	81
accuracy			0.43	4624
macro avg	0.59	0.48	0.47	4624
weighted avg	0.59	0.43	0.46	4624

kNN with actual labels:

	precision	recall	f1-score	support
0	0.37	0.63	0.46	200
1	0.25	0.51	0.33	200
2	0.31	0.59	0.41	200
3	0.51	0.37	0.43	200
4	0.27	0.60	0.37	200
5	0.23	0.54	0.32	200
6	0.38	0.58	0.46	200
7	0.70	0.49	0.58	200
8	0.64	0.47	0.54	200
9	0.82	0.48	0.61	200
10	0.78	0.58	0.67	200
11	0.92	0.50	0.65	200
12	0.63	0.41	0.50	200
13	0.81	0.41	0.55	200
14	0.76	0.57	0.65	200
15	0.89	0.91	0.90	200
16	0.68	0.57	0.62	200
17	0.74	0.72	0.73	200
18	0.76	0.42	0.54	200
19	0.52	0.36	0.42	200
20	0.90	0.44	0.59	204
21	0.96	0.41	0.58	155
22	1.00	0.32	0.49	102
23	0.98	0.51	0.67	83
24	1.00	0.30	0.46	80
accuracy			0.52	4624
macro avg	0.67	0.51	0.54	4624
weighted avg	0.65	0.52	0.54	4624

MultinomialNB with predicted labels:

	precision	recall	f1-score	support
0	0.65	0.78	0.71	173
1	0.30	0.86	0.44	81
2	0.06	0.58	0.10	12
3	0.60	0.68	0.64	194
4	0.64	0.73	0.68	185
5	0.62	0.61	0.61	213
6	0.96	0.35	0.51	512
7	0.66	0.83	0.73	155
8	0.87	0.47	0.61	370
9	0.73	0.82	0.78	171
10	0.69	0.96	0.81	145
11	0.64	0.97	0.77	139
12	0.82	0.31	0.45	515
13	0.68	0.86	0.76	151
14	0.61	0.96	0.75	124
15	0.90	0.87	0.88	220
16	0.65	0.76	0.70	188
17	0.67	1.00	0.80	135
18	0.59	0.71	0.64	199
19	0.48	0.64	0.55	120
20	0.92	0.97	0.95	199
21	0.91	0.99	0.95	140
22	0.97	0.97	0.97	101
23	1.00	0.80	0.89	101
24	0.97	0.94	0.96	81
accuracy			0.69	4624
macro avg	0.70	0.78	0.71	4624
eighted avg	0.76	0.69	0.68	4624

MultinomialNB with actual labels:

	precision	recall	f1-score	support
0	0.78	0.81	0.80	200
1	0.74	0.88	0.80	200
2	0.91	0.57	0.71	200
3	0.73	0.81	0.77	200
4	0.86	0.90	0.88	200
5	0.81	0.85	0.83	200
6	0.86	0.80	0.83	200
7	0.93	0.91	0.92	200
8	0.96	0.96	0.96	200
9	0.98	0.94	0.96	200
10	0.97	0.97	0.97	200
11	0.86	0.92	0.89	200
12	0.86	0.83	0.84	200
13	0.95	0.91	0.93	200
14	0.94	0.91	0.92	200
15	0.93	0.98	0.96	200
16	0.79	0.86	0.82	200
17	0.93	0.93	0.93	200
18	0.66	0.80	0.72	200
19	0.68	0.55	0.61	200
20	0.97	1.00	0.98	204
21	0.99	0.97	0.98	155
22	1.00	0.99	1.00	102
23	0.98	0.95	0.96	83
24	0.96	0.94	0.95	80
accuracy			0.87	4624
macro avg	0.88	0.88	0.88	4624
weighted avg	0.87	0.87	0.87	4624

SGDClassifier with predicted labels:

	precision	recall	f1-score	support
0	0.60	0.73	0.66	173
1	0.27	0.78	0.40	81
2	0.07	1.00	0.12	12
3	0.55	0.54	0.54	194
4	0.66	0.72	0.69	185
5	0.59	0.54	0.56	213
6	0.88	0.35	0.50	512
7	0.61	0.83	0.70	155
8	0.90	0.44	0.59	370
9	0.77	0.82	0.79	171
10	0.71	0.94	0.81	145
11	0.66	0.94	0.78	139
12	0.73	0.27	0.39	515
13	0.62	0.83	0.71	151
14	0.58	0.97	0.73	124
15	0.91	0.83	0.86	220
16	0.62	0.69	0.65	188
17	0.69	0.96	0.80	135
18	0.54	0.61	0.57	199
19	0.34	0.56	0.42	120
20	0.93	0.94	0.94	199
21	0.90	0.99	0.94	140
22	0.98	0.93	0.95	101
23	0.94	0.80	0.87	101
24	0.94	0.93	0.93	81
accuracy			0.65	4624
macro avg	0.68	0.76	0.68	4624
weighted avg	0.73	0.65	0.65	4624

SGDClassifier with actual labels:

	precision	recall	f1-score	support
0	0.73	0.76	0.75	200
1	0.69	0.80	0.74	200
2	0.74	0.77	0.75	200
3	0.75	0.66	0.70	200
4	0.81	0.85	0.83	200
5	0.89	0.78	0.83	200
6	0.83	0.81	0.82	200
7	0.90	0.89	0.89	200
8	0.96	0.91	0.93	200
9	0.95	0.90	0.92	200
10	0.95	0.95	0.95	200
11	0.92	0.92	0.92	200
12	0.79	0.77	0.78	200
13	0.84	0.89	0.86	200
14	0.88	0.91	0.89	200
15	0.99	0.96	0.98	200
16	0.79	0.81	0.80	200
17	0.92	0.88	0.90	200
18	0.61	0.72	0.67	200
19	0.54	0.54	0.54	200
20	0.99	0.96	0.98	204
21	1.00	0.99	0.99	155
22	0.98	0.98	0.98	102
23	0.93	0.94	0.93	83
24	1.00	0.97	0.99	80
accuracy			0.84	4624
macro avg	0.86	0.85	0.85	4624
weighted avg	0.85	0.84	0.84	4624

LogisticRegression with predicted labels:

	precision	recall	f1-score	support
0	0.60	0.72	0.65	173
1	0.29	0.75	0.42	81
2	0.04	0.67	0.08	12
3	0.55	0.56	0.56	194
4	0.67	0.72	0.69	185
5	0.56	0.54	0.55	213
6	0.92	0.38	0.53	512
7	0.63	0.83	0.71	155
8	0.89	0.46	0.61	370
9	0.74	0.82	0.78	171
10	0.70	0.97	0.82	145
11	0.67	0.96	0.79	139
12	0.78	0.31	0.44	515
13	0.66	0.87	0.75	151
14	0.63	0.94	0.75	124
15	0.91	0.82	0.86	220
16	0.66	0.72	0.69	188
17	0.66	0.98	0.79	135
18	0.55	0.57	0.56	199
19	0.37	0.57	0.45	120
20	0.95	0.98	0.96	199
21	0.89	0.99	0.94	140
22	0.98	0.98	0.98	101
23	1.00	0.79	0.88	101
24	0.95	0.93	0.94	81
accuracy			0.67	4624
macro avg	0.69	0.75	0.69	4624
weighted avg	0.74	0.67	0.67	4624

LogisticRegression with actual labels:

	precision	recall	f1-score	support
0	0.77	0.80	0.78	200
1	0.79	0.83	0.81	200
2	0.82	0.80	0.81	200
3	0.76	0.74	0.75	200
4	0.87	0.87	0.87	200
5	0.82	0.84	0.83	200
6	0.84	0.88	0.86	200
7	0.88	0.91	0.90	200
8	0.94	0.92	0.93	200
9	0.95	0.90	0.92	200
10	0.96	0.96	0.96	200
11	0.94	0.94	0.94	200
12	0.83	0.84	0.83	200
13	0.90	0.90	0.90	200
14	0.96	0.89	0.92	200
15	0.99	0.99	0.99	200
16	0.81	0.83	0.82	200
17	0.92	0.92	0.92	200
18	0.71	0.73	0.72	200
19	0.60	0.56	0.58	200
20	0.99	1.00	1.00	204
21	1.00	1.00	1.00	155
22	1.00	0.99	1.00	102
23	1.00	0.96	0.98	83
24	1.00	0.99	0.99	80
accuracy			0.87	4624
macro avg	0.88	0.88	0.88	4624
weighted avg	0.87	0.87	0.87	4624

With train – 30% and test – 70%:
Accuracy between actual and predicted labels:

```
accuracy = accuracy_score(y_test, test_class_predictions)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.6479021195081258
```

kNN with predicted labels:

	precision	recall	f1-score	support
0	0.51	0.49	0.50	577
1	0.11	0.49	0.18	282
2	0.01	0.62	0.03	55
3	0.41	0.20	0.26	781
4	0.37	0.20	0.26	827
5	0.07	0.41	0.11	604
6	0.29	0.32	0.30	1249
7	0.47	0.34	0.39	582
8	0.35	0.19	0.25	1352
9	0.76	0.32	0.45	515
10	0.68	0.50	0.57	571
11	0.76	0.47	0.58	494
12	0.51	0.07	0.12	2161
13	0.43	0.42	0.43	435
14	0.63	0.31	0.41	429
15	0.90	0.66	0.77	779
16	0.74	0.38	0.50	625
17	0.46	0.80	0.58	411
18	0.75	0.28	0.41	745
19	0.44	0.28	0.34	524
20	0.93	0.30	0.45	711
21	0.94	0.30	0.46	486
22	0.97	0.19	0.31	362
23	0.96	0.35	0.51	345
24	1.00	0.15	0.27	281
accuracy			0.31	16183
macro avg	0.58	0.36	0.38	16183
weighted avg	0.56	0.31	0.36	16183

kNN with actual labels:

	precision	recall	f1-score	support
0	0.58	0.45	0.51	700
1	0.21	0.38	0.27	700
2	0.15	0.54	0.23	700
3	0.41	0.22	0.28	700
4	0.40	0.26	0.31	700
5	0.10	0.54	0.17	700
6	0.25	0.47	0.32	700
7	0.48	0.29	0.36	700
8	0.32	0.34	0.33	700
9	0.79	0.24	0.37	700
10	0.71	0.42	0.53	700
11	0.87	0.38	0.53	700
12	0.48	0.20	0.28	700
13	0.54	0.33	0.41	700
14	0.80	0.24	0.37	700
15	0.89	0.73	0.80	698
16	0.72	0.33	0.46	700
17	0.56	0.57	0.56	700
18	0.75	0.30	0.43	700
19	0.53	0.25	0.34	700
20	0.96	0.31	0.47	714
21	0.94	0.27	0.42	540
22	0.97	0.19	0.32	358
23	0.97	0.41	0.58	292
24	1.00	0.15	0.27	281
accuracy			0.36	16183
macro avg	0.61	0.35	0.40	16183
weighted avg	0.59	0.36	0.40	16183

MultinomialNB with predicted labels:

	precision	recall	f1-score	support
0	0.70	0.70	0.70	577
1	0.21	0.89	0.34	282
2	0.14	0.71	0.24	55
3	0.67	0.72	0.69	781
4	0.78	0.55	0.65	827
5	0.57	0.68	0.62	604
6	0.95	0.38	0.54	1249
7	0.67	0.82	0.74	582
8	0.90	0.42	0.58	1352
9	0.70	0.92	0.80	515
10	0.78	0.96	0.86	571
11	0.57	0.96	0.72	494
12	0.86	0.22	0.35	2161
13	0.58	0.89	0.71	435
14	0.54	0.95	0.69	429
15	0.87	0.90	0.89	779
16	0.66	0.81	0.73	625
17	0.50	1.00	0.67	411
18	0.54	0.63	0.58	745
19	0.60	0.64	0.62	524
20	0.90	0.95	0.92	711
21	0.88	0.99	0.93	486
22	0.99	0.96	0.98	362
23	0.99	0.72	0.83	345
24	0.94	0.87	0.90	281
accuracy			0.67	16183
macro avg	0.70	0.77	0.69	16183
weighted avg	0.76	0.67	0.66	16183

MultinomialNB with actual labels:

	precision	recall	f1-score	support
0	0.80	0.66	0.73	700
1	0.51	0.87	0.64	700
2	0.92	0.35	0.51	700
3	0.64	0.77	0.70	700
4	0.89	0.75	0.81	700
5	0.75	0.77	0.76	700
6	0.90	0.64	0.75	700
7	0.86	0.87	0.87	700
8	0.96	0.88	0.92	700
9	0.97	0.93	0.95	700
10	0.96	0.97	0.96	700
11	0.78	0.93	0.85	700
12	0.87	0.67	0.75	700
13	0.94	0.89	0.91	700
14	0.86	0.93	0.89	700
15	0.85	0.99	0.92	698
16	0.77	0.84	0.80	700
17	0.81	0.94	0.87	700
18	0.61	0.76	0.68	700
19	0.63	0.50	0.56	700
20	0.95	0.99	0.97	714
21	0.98	0.98	0.98	540
22	1.00	0.99	0.99	358
23	0.98	0.84	0.90	292
24	0.99	0.91	0.95	281
accuracy			0.82	16183
macro avg	0.85	0.82	0.82	16183
weighted avg	0.84	0.82	0.81	16183

SGDClassifier with predicted labels:

	precision	recall	f1-score	support
0	0.62	0.66	0.64	577
1	0.24	0.68	0.36	282
2	0.07	0.75	0.12	55
3	0.63	0.58	0.60	781
4	0.71	0.58	0.64	827
5	0.54	0.54	0.54	604
6	0.83	0.47	0.60	1249
7	0.68	0.84	0.75	582
8	0.90	0.40	0.56	1352
9	0.70	0.90	0.79	515
10	0.68	0.96	0.79	571
11	0.65	0.92	0.76	494
12	0.82	0.24	0.38	2161
13	0.55	0.89	0.68	435
14	0.53	0.97	0.68	429
15	0.91	0.81	0.86	779
16	0.49	0.81	0.61	625
17	0.50	0.97	0.66	411
18	0.54	0.57	0.55	745
19	0.59	0.47	0.52	524
20	0.94	0.93	0.93	711
21	0.91	0.98	0.94	486
22	0.97	0.96	0.96	362
23	0.98	0.78	0.87	345
24	0.93	0.90	0.91	281
accuracy			0.65	16183
macro avg	0.68	0.74	0.67	16183
weighted avg	0.73	0.65	0.64	16183

SGDClassifier with actual labels:

	precision	recall	f1-score	support
0	0.79	0.61	0.69	700
1	0.70	0.63	0.66	700
2	0.70	0.62	0.66	700
3	0.54	0.75	0.63	700
4	0.84	0.71	0.77	700
5	0.71	0.70	0.71	700
6	0.69	0.77	0.73	700
7	0.74	0.78	0.76	700
8	0.85	0.85	0.85	700
9	0.90	0.85	0.87	700
10	0.78	0.94	0.85	700
11	0.89	0.85	0.87	700
12	0.71	0.69	0.70	700
13	0.89	0.77	0.82	700
14	0.75	0.89	0.82	700
15	0.95	0.97	0.96	698
16	0.78	0.76	0.77	700
17	0.88	0.84	0.86	700
18	0.60	0.63	0.62	700
19	0.55	0.57	0.56	700
20	0.96	0.94	0.95	714
21	0.96	0.95	0.95	540
22	0.97	0.97	0.97	358
23	0.96	0.90	0.93	292
24	0.95	0.86	0.90	281
accuracy			0.78	16183
macro avg	0.80	0.79	0.79	16183
weighted avg	0.79	0.78	0.78	16183

LogisticRegression with predicted labels:

	precision	recall	f1-score	support
0	0.62	0.71	0.66	577
1	0.26	0.70	0.38	282
2	0.06	0.75	0.11	55
3	0.62	0.57	0.59	781
4	0.73	0.64	0.68	827
5	0.50	0.56	0.53	604
6	0.84	0.53	0.65	1249
7	0.74	0.87	0.80	582
8	0.91	0.45	0.61	1352
9	0.72	0.93	0.81	515
10	0.77	0.96	0.85	571
11	0.67	0.94	0.78	494
12	0.85	0.29	0.43	2161
13	0.58	0.90	0.71	435
14	0.61	0.97	0.75	429
15	0.91	0.83	0.87	779
16	0.68	0.76	0.72	625
17	0.61	0.96	0.74	411
18	0.58	0.59	0.59	745
19	0.50	0.66	0.57	524
20	0.94	0.93	0.93	711
21	0.89	0.99	0.93	486
22	0.97	0.98	0.98	362
23	0.98	0.80	0.88	345
24	0.97	0.83	0.89	281
accuracy			0.68	16183
macro avg	0.70	0.76	0.70	16183
weighted avg	0.75	0.68	0.68	16183

LogisticRegression with actual labels:

	precision	recall	f1-score	support
0	0.78	0.73	0.75	700
1	0.70	0.75	0.72	700
2	0.76	0.75	0.76	700
3	0.69	0.71	0.70	700
4	0.79	0.81	0.80	700
5	0.76	0.73	0.74	700
6	0.76	0.85	0.80	700
7	0.85	0.84	0.85	700
8	0.94	0.90	0.92	700
9	0.94	0.90	0.92	700
10	0.93	0.95	0.94	700
11	0.91	0.90	0.91	700
12	0.74	0.78	0.76	700
13	0.89	0.87	0.88	700
14	0.91	0.88	0.90	700
15	0.97	0.98	0.98	698
16	0.80	0.80	0.80	700
17	0.93	0.86	0.89	700
18	0.65	0.70	0.67	700
19	0.61	0.60	0.60	700
20	0.97	0.96	0.96	714
21	0.97	0.97	0.97	540
22	0.97	0.99	0.98	358
23	0.94	0.91	0.92	292
24	0.97	0.83	0.89	281
accuracy			0.83	16183
macro avg	0.84	0.84	0.84	16183
weighted avg	0.84	0.83	0.83	16183

CountVectorizer with n-grams:
Train – 80% and test – 20%
Accuracy between actual and predicted labels:

```
accuracy = accuracy_score(y_test, test_class_predictions)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.6563581314878892
```

kNN with predicted labels:

	precision	recall	f1-score	support
0	0.50	0.62	0.56	177
1	0.05	0.49	0.09	77
2	0.02	0.57	0.03	7
3	0.35	0.33	0.34	197
4	0.24	0.35	0.29	200
5	0.27	0.32	0.29	201
6	0.45	0.22	0.29	490
7	0.39	0.51	0.45	154
8	0.57	0.26	0.36	397
9	0.46	0.55	0.50	166
10	0.64	0.58	0.61	142
11	0.76	0.54	0.64	136
12	0.54	0.14	0.22	522
13	0.50	0.46	0.48	147
14	0.62	0.65	0.63	131
15	0.93	0.82	0.87	218
16	0.74	0.53	0.62	194
17	0.43	0.79	0.56	131
18	0.48	0.32	0.39	191
19	0.43	0.46	0.44	122
20	0.91	0.68	0.78	202
21	0.85	0.60	0.70	140
22	0.90	0.74	0.82	101
23	1.00	0.51	0.67	99
24	0.96	0.33	0.49	82
accuracy			0.43	4624
macro avg	0.56	0.50	0.48	4624
weighted avg	0.56	0.43	0.46	4624

kNN with actual labels:

	precision	recall	f1-score	support
0	0.56	0.61	0.59	200
1	0.13	0.49	0.21	200
2	0.40	0.52	0.45	200
3	0.40	0.38	0.39	200
4	0.28	0.41	0.33	200
5	0.37	0.44	0.40	200
6	0.39	0.46	0.42	200
7	0.49	0.49	0.49	200
8	0.54	0.49	0.51	200
9	0.50	0.49	0.50	200
10	0.77	0.50	0.61	200
11	0.91	0.44	0.59	200
12	0.54	0.36	0.44	200
13	0.62	0.42	0.50	200
14	0.81	0.56	0.66	200
15	0.92	0.89	0.90	200
16	0.76	0.53	0.63	200
17	0.57	0.69	0.62	200
18	0.60	0.39	0.47	200
19	0.55	0.36	0.44	200
20	0.92	0.69	0.79	204
21	0.92	0.59	0.72	155
22	0.90	0.74	0.81	102
23	0.98	0.59	0.74	83
24	0.96	0.34	0.50	80
accuracy			0.51	4624
macro avg	0.63	0.51	0.55	4624
weighted avg	0.61	0.51	0.54	4624

MultinomialNB with predicted labels:

	precision	recall	f1-score	support
0	0.73	0.83	0.78	177
1	0.34	0.78	0.47	77
2	0.03	0.57	0.05	7
3	0.54	0.73	0.62	197
4	0.66	0.69	0.67	200
5	0.62	0.61	0.61	201
6	0.92	0.48	0.63	490
7	0.63	0.84	0.72	154
8	0.80	0.47	0.59	397
9	0.74	0.86	0.79	166
10	0.72	0.96	0.82	142
11	0.67	0.96	0.79	136
12	0.82	0.33	0.47	522
13	0.68	0.83	0.75	147
14	0.68	0.94	0.79	131
15	0.96	0.94	0.95	218
16	0.73	0.85	0.79	194
17	0.70	0.99	0.82	131
18	0.70	0.65	0.68	191
19	0.54	0.70	0.61	122
20	0.97	0.96	0.97	202
21	0.92	0.99	0.96	140
22	0.96	0.97	0.97	101
23	0.97	0.85	0.90	99
24	0.94	0.96	0.95	82
accuracy			0.71	4624
macro avg	0.72	0.79	0.73	4624
weighted avg	0.77	0.71	0.71	4624

MultinomialNB with actual labels:

	precision	recall	f1-score	support
0	0.79	0.79	0.79	200
1	0.78	0.69	0.73	200
2	0.81	0.61	0.70	200
3	0.61	0.81	0.70	200
4	0.79	0.82	0.80	200
5	0.81	0.80	0.80	200
6	0.70	0.89	0.78	200
7	0.85	0.88	0.87	200
8	0.80	0.94	0.87	200
9	0.91	0.88	0.90	200
10	0.96	0.92	0.94	200
11	0.93	0.91	0.92	200
12	0.74	0.78	0.76	200
13	0.94	0.84	0.89	200
14	0.96	0.87	0.91	200
15	0.93	0.99	0.96	200
16	0.75	0.84	0.79	200
17	0.95	0.89	0.91	200
18	0.75	0.66	0.70	200
19	0.68	0.54	0.60	200
20	0.99	0.97	0.98	204
21	0.99	0.96	0.97	155
22	0.99	0.99	0.99	102
23	0.93	0.98	0.95	83
24	0.95	1.00	0.98	80
accuracy			0.84	4624
macro avg	0.85	0.85	0.85	4624
weighted avg	0.84	0.84	0.84	4624

SGDClassifier with predicted labels:

	precision	recall	f1-score	support
0	0.66	0.76	0.71	177
1	0.25	0.71	0.37	77
2	0.03	1.00	0.07	7
3	0.59	0.55	0.57	197
4	0.63	0.62	0.63	200
5	0.50	0.54	0.52	201
6	0.88	0.38	0.53	490
7	0.67	0.79	0.73	154
8	0.86	0.42	0.56	397
9	0.74	0.81	0.77	166
10	0.68	0.96	0.80	142
11	0.65	0.96	0.77	136
12	0.67	0.27	0.39	522
13	0.65	0.81	0.72	147
14	0.64	0.92	0.75	131
15	0.98	0.90	0.94	218
16	0.67	0.71	0.69	194
17	0.54	0.95	0.69	131
18	0.55	0.60	0.58	191
19	0.43	0.61	0.50	122
20	0.94	0.92	0.93	202
21	0.90	0.99	0.94	140
22	0.95	0.95	0.95	101
23	0.95	0.81	0.87	99
24	0.93	0.91	0.92	82
accuracy			0.65	4624
macro avg	0.68	0.75	0.68	4624
weighted avg	0.72	0.65	0.65	4624

SGDClassifier with actual labels:

	precision	recall	f1-score	support
0	0.74	0.80	0.77	200
1	0.70	0.77	0.74	200
2	0.77	0.72	0.75	200
3	0.73	0.69	0.71	200
4	0.81	0.80	0.80	200
5	0.84	0.81	0.83	200
6	0.84	0.81	0.82	200
7	0.79	0.89	0.83	200
8	0.93	0.89	0.91	200
9	0.92	0.93	0.92	200
10	0.95	0.95	0.95	200
11	0.92	0.90	0.91	200
12	0.76	0.73	0.75	200
13	0.84	0.86	0.85	200
14	0.91	0.89	0.90	200
15	0.98	1.00	0.99	200
16	0.79	0.82	0.81	200
17	0.87	0.89	0.88	200
18	0.63	0.69	0.66	200
19	0.58	0.48	0.53	200
20	0.99	0.98	0.99	204
21	1.00	0.99	0.99	155
22	1.00	0.97	0.99	102
23	0.91	0.96	0.94	83
24	1.00	0.99	0.99	80
accuracy			0.84	4624
macro avg	0.85	0.85	0.85	4624
weighted avg	0.84	0.84	0.84	4624

LogisticRegression with predicted labels:

	precision	recall	f1-score	support
0	0.66	0.75	0.70	177
1	0.28	0.74	0.40	77
2	0.02	0.57	0.04	7
3	0.54	0.60	0.57	197
4	0.66	0.63	0.64	200
5	0.54	0.54	0.54	201
6	0.91	0.39	0.54	490
7	0.64	0.86	0.74	154
8	0.87	0.42	0.57	397
9	0.71	0.84	0.77	166
10	0.70	0.96	0.81	142
11	0.66	0.96	0.78	136
12	0.75	0.29	0.42	522
13	0.61	0.85	0.71	147
14	0.65	0.94	0.77	131
15	0.99	0.91	0.95	218
16	0.71	0.73	0.72	194
17	0.65	0.98	0.78	131
18	0.58	0.61	0.59	191
19	0.41	0.63	0.49	122
20	0.94	0.97	0.96	202
21	0.89	0.99	0.94	140
22	0.98	0.98	0.98	101
23	0.99	0.80	0.88	99
24	0.95	0.93	0.94	82
accuracy			0.67	4624
macro avg	0.69	0.75	0.69	4624
weighted avg	0.74	0.67	0.67	4624

LogisticRegression with actual labels:

	precision	recall	f1-score	support
0	0.77	0.79	0.78	200
1	0.74	0.77	0.75	200
2	0.80	0.79	0.79	200
3	0.70	0.77	0.74	200
4	0.87	0.83	0.85	200
5	0.82	0.83	0.82	200
6	0.83	0.85	0.84	200
7	0.86	0.89	0.87	200
8	0.94	0.91	0.92	200
9	0.90	0.89	0.90	200
10	0.97	0.95	0.96	200
11	0.94	0.94	0.94	200
12	0.80	0.80	0.80	200
13	0.87	0.89	0.88	200
14	0.94	0.89	0.92	200
15	0.99	0.99	0.99	200
16	0.82	0.81	0.81	200
17	0.92	0.91	0.92	200
18	0.70	0.70	0.70	200
19	0.59	0.56	0.57	200
20	0.98	1.00	0.99	204
21	1.00	1.00	1.00	155
22	1.00	0.99	1.00	102
23	1.00	0.96	0.98	83
24	1.00	1.00	1.00	80
accuracy			0.86	4624
macro avg	0.87	0.87	0.87	4624
weighted avg	0.86	0.86	0.86	4624

With train – 30% and test – 70%:
Accuracy between actual and predicted labels:

```
accuracy = accuracy_score(y_test, test_class_predictions)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.646171908793178
```

kNN with predicted labels:

	precision	recall	f1-score	support
0	0.45	0.54	0.49	548
1	0.06	0.55	0.10	278
2	0.01	0.44	0.01	18
3	0.36	0.22	0.28	769
4	0.33	0.18	0.23	857
5	0.17	0.31	0.22	616
6	0.26	0.30	0.28	1257
7	0.26	0.38	0.31	564
8	0.39	0.26	0.31	1457
9	0.36	0.42	0.39	499
10	0.53	0.50	0.51	559
11	0.80	0.45	0.58	494
12	0.35	0.11	0.17	2123
13	0.50	0.43	0.46	451
14	0.51	0.45	0.48	423
15	0.96	0.76	0.85	793
16	0.77	0.38	0.51	634
17	0.35	0.79	0.49	387
18	0.77	0.24	0.37	732
19	0.53	0.32	0.40	533
20	0.89	0.54	0.67	715
21	0.88	0.48	0.62	484
22	0.87	0.52	0.66	368
23	0.96	0.44	0.60	339
24	1.00	0.23	0.37	285
accuracy			0.36	16183
macro avg	0.53	0.41	0.42	16183
eighted avg	0.51	0.36	0.39	16183

kNN with actual labels:

	precision	recall	f1-score	support
0	0.51	0.48	0.50	700
1	0.12	0.48	0.19	700
2	0.23	0.38	0.29	700
3	0.35	0.24	0.28	700
4	0.35	0.23	0.28	700
5	0.22	0.34	0.27	700
6	0.21	0.42	0.28	700
7	0.29	0.35	0.32	700
8	0.29	0.39	0.33	700
9	0.44	0.36	0.39	700
10	0.57	0.43	0.49	700
11	0.89	0.36	0.51	700
12	0.27	0.27	0.27	700
13	0.63	0.35	0.45	700
14	0.62	0.34	0.44	700
15	0.93	0.84	0.89	698
16	0.72	0.32	0.44	700
17	0.44	0.55	0.49	700
18	0.76	0.25	0.38	700
19	0.56	0.26	0.36	700
20	0.90	0.55	0.68	714
21	0.92	0.44	0.60	540
22	0.86	0.53	0.66	358
23	0.94	0.50	0.65	292
24	1.00	0.23	0.38	281
accuracy			0.39	16183
macro avg	0.56	0.40	0.43	16183
weighted avg	0.53	0.39	0.42	16183

MultinomialNB with predicted labels:

	precision	recall	f1-score	support
0	0.67	0.83	0.74	548
1	0.38	0.76	0.50	278
2	0.02	0.67	0.04	18
3	0.60	0.74	0.66	769
4	0.66	0.65	0.66	857
5	0.54	0.60	0.57	616
6	0.86	0.56	0.68	1257
7	0.65	0.83	0.73	564
8	0.81	0.47	0.59	1457
9	0.68	0.92	0.79	499
10	0.78	0.96	0.86	559
11	0.70	0.94	0.80	494
12	0.85	0.32	0.46	2123
13	0.67	0.84	0.74	451
14	0.64	0.95	0.76	423
15	0.95	0.91	0.93	793
16	0.73	0.88	0.80	634
17	0.62	0.99	0.76	387
18	0.69	0.60	0.64	732
19	0.64	0.68	0.66	533
20	0.97	0.93	0.95	715
21	0.88	0.99	0.93	484
22	0.99	0.96	0.97	368
23	0.96	0.86	0.91	339
24	0.92	0.95	0.94	285
accuracy			0.71	16183
macro avg	0.71	0.79	0.72	16183
weighted avg	0.76	0.71	0.71	16183

MultinomialNB with actual labels:

	precision	recall	f1-score	support
0	0.77	0.76	0.76	700
1	0.74	0.59	0.66	700
2	0.78	0.60	0.68	700
3	0.58	0.78	0.66	700
4	0.67	0.82	0.74	700
5	0.81	0.78	0.79	700
6	0.74	0.86	0.79	700
7	0.82	0.84	0.83	700
8	0.75	0.91	0.82	700
9	0.93	0.90	0.92	700
10	0.96	0.94	0.95	700
11	0.93	0.88	0.91	700
12	0.67	0.75	0.70	700
13	0.95	0.77	0.85	700
14	0.95	0.85	0.89	700
15	0.91	0.99	0.95	698
16	0.75	0.82	0.79	700
17	0.93	0.83	0.88	700
18	0.73	0.67	0.70	700
19	0.65	0.52	0.58	700
20	0.98	0.95	0.97	714
21	0.98	0.99	0.98	540
22	0.99	0.99	0.99	358
23	0.93	0.96	0.94	292
24	0.92	0.96	0.94	281
accuracy			0.82	16183
macro avg	0.83	0.83	0.83	16183
weighted avg	0.82	0.82	0.82	16183

SGDClassifier with predicted labels:

	precision	recall	f1-score	support
0	0.61	0.70	0.65	548
1	0.24	0.65	0.35	278
2	0.03	1.00	0.05	18
3	0.62	0.55	0.59	769
4	0.71	0.54	0.61	857
5	0.55	0.52	0.53	616
6	0.81	0.46	0.58	1257
7	0.57	0.83	0.67	564
8	0.82	0.42	0.55	1457
9	0.70	0.90	0.79	499
10	0.76	0.94	0.84	559
11	0.70	0.93	0.80	494
12	0.79	0.26	0.39	2123
13	0.52	0.84	0.64	451
14	0.50	0.97	0.66	423
15	0.99	0.86	0.92	793
16	0.76	0.63	0.69	634
17	0.60	0.96	0.73	387
18	0.53	0.56	0.54	732
19	0.37	0.70	0.49	533
20	0.93	0.94	0.93	715
21	0.90	0.98	0.94	484
22	0.99	0.91	0.95	368
23	0.97	0.78	0.86	339
24	0.95	0.86	0.90	285
accuracy			0.64	16183
macro avg	0.68	0.75	0.67	16183
weighted avg	0.73	0.64	0.65	16183

SGDClassifier with actual labels:

	precision	recall	f1-score	support
0	0.70	0.70	0.70	700
1	0.60	0.69	0.64	700
2	0.70	0.62	0.66	700
3	0.69	0.61	0.65	700
4	0.78	0.70	0.74	700
5	0.70	0.69	0.70	700
6	0.73	0.76	0.75	700
7	0.72	0.78	0.75	700
8	0.84	0.84	0.84	700
9	0.93	0.78	0.85	700
10	0.83	0.94	0.88	700
11	0.88	0.87	0.88	700
12	0.56	0.76	0.64	700
13	0.81	0.80	0.80	700
14	0.79	0.86	0.82	700
15	0.96	0.98	0.97	698
16	0.75	0.78	0.77	700
17	0.89	0.87	0.88	700
18	0.60	0.62	0.61	700
19	0.54	0.49	0.51	700
20	0.97	0.92	0.95	714
21	0.97	0.95	0.96	540
22	0.98	0.97	0.98	358
23	0.89	0.92	0.91	292
24	0.99	0.32	0.48	281
accuracy			0.77	16183
macro avg	0.79	0.77	0.77	16183
weighted avg	0.78	0.77	0.77	16183

LogisticRegression with predicted labels:

	precision	recall	f1-score	support
0	0.63	0.77	0.69	548
1	0.23	0.65	0.34	278
2	0.01	0.50	0.03	18
3	0.60	0.57	0.58	769
4	0.71	0.61	0.66	857
5	0.50	0.54	0.52	616
6	0.86	0.52	0.65	1257
7	0.69	0.86	0.77	564
8	0.89	0.42	0.57	1457
9	0.68	0.92	0.78	499
10	0.75	0.96	0.84	559
11	0.68	0.95	0.79	494
12	0.81	0.29	0.42	2123
13	0.56	0.85	0.68	451
14	0.62	0.96	0.75	423
15	0.98	0.87	0.93	793
16	0.72	0.77	0.74	634
17	0.57	0.96	0.71	387
18	0.60	0.60	0.60	732
19	0.54	0.69	0.61	533
20	0.94	0.93	0.94	715
21	0.88	0.98	0.93	484
22	0.97	0.98	0.97	368
23	0.98	0.83	0.90	339
24	0.97	0.84	0.90	285
accuracy			0.67	16183
macro avg	0.70	0.75	0.69	16183
weighted avg	0.75	0.67	0.68	16183

LogisticRegression with actual labels:

	precision	recall	f1-score	support
0	0.77	0.74	0.75	700
1	0.64	0.71	0.67	700
2	0.74	0.73	0.74	700
3	0.67	0.70	0.69	700
4	0.74	0.79	0.77	700
5	0.76	0.71	0.74	700
6	0.78	0.84	0.81	700
7	0.80	0.81	0.81	700
8	0.88	0.88	0.88	700
9	0.90	0.87	0.89	700
10	0.92	0.94	0.93	700
11	0.92	0.90	0.91	700
12	0.68	0.73	0.71	700
13	0.85	0.82	0.83	700
14	0.91	0.86	0.88	700
15	0.98	0.99	0.99	698
16	0.81	0.79	0.80	700
17	0.92	0.86	0.89	700
18	0.66	0.69	0.68	700
19	0.62	0.60	0.61	700
20	0.97	0.96	0.96	714
21	0.97	0.97	0.97	540
22	0.96	1.00	0.98	358
23	0.94	0.92	0.93	292
24	0.97	0.85	0.90	281
accuracy			0.82	16183
macro avg	0.83	0.83	0.83	16183
weighted avg	0.82	0.82	0.82	16183

With unlabeled dataset:
kNN:

	precision	recall	f1-score	support
0	0.48	0.62	0.54	257
1	0.34	0.52	0.41	295
2	0.27	0.63	0.37	206
3	0.51	0.36	0.42	227
4	0.24	0.54	0.33	230
5	0.20	0.68	0.30	401
6	0.49	0.40	0.44	272
7	0.71	0.47	0.56	219
8	0.71	0.51	0.59	220
9	0.69	0.45	0.54	299
10	0.73	0.51	0.60	309
11	0.71	0.54	0.62	425
12	0.58	0.39	0.47	235
13	0.66	0.40	0.50	290
14	0.79	0.55	0.65	301
15	0.91	0.63	0.75	278
16	0.78	0.46	0.58	274
17	0.73	0.61	0.66	370
18	0.67	0.40	0.50	295
19	0.61	0.45	0.52	280
20	0.79	0.50	0.61	334
21	0.70	0.47	0.56	352
22	0.65	0.45	0.54	343
23	0.72	0.45	0.55	175
24	0.67	0.32	0.43	161
accuracy			0.50	7048
macro avg	0.61	0.49	0.52	7048
weighted avg	0.62	0.50	0.53	7048

MultinomialNB:

	precision	recall	f1-score	support
0	0.76	0.74	0.75	257
1	0.76	0.67	0.72	295
2	0.75	0.78	0.77	206
3	0.70	0.79	0.74	227
4	0.78	0.83	0.81	230
5	0.81	0.72	0.76	401
6	0.83	0.73	0.77	272
7	0.91	0.87	0.89	219
8	0.90	0.89	0.89	220
9	0.94	0.76	0.84	299
10	0.97	0.75	0.85	309
11	0.91	0.64	0.75	425
12	0.85	0.77	0.81	235
13	0.96	0.72	0.83	290
14	0.98	0.79	0.88	301
15	0.95	0.73	0.83	278
16	0.79	0.75	0.77	274
17	0.98	0.61	0.75	370
18	0.80	0.55	0.65	295
19	0.77	0.50	0.61	280
20	0.99	0.67	0.80	334
21	0.73	0.80	0.76	352
22	0.25	1.00	0.40	343
23	0.92	0.77	0.84	175
24	0.95	0.66	0.78	161
accuracy			0.74	7048
macro avg	0.84	0.74	0.77	7048
weighted avg	0.83	0.74	0.76	7048

SGDClassifier:

	precision	recall	f1-score	support
0	0.81	0.74	0.77	257
1	0.71	0.80	0.75	295
2	0.72	0.76	0.74	206
3	0.81	0.73	0.77	227
4	0.85	0.86	0.85	230
5	0.82	0.85	0.83	401
6	0.84	0.84	0.84	272
7	0.91	0.89	0.90	219
8	0.91	0.90	0.91	220
9	0.93	0.92	0.93	299
10	0.92	0.91	0.91	309
11	0.89	0.89	0.89	425
12	0.81	0.82	0.81	235
13	0.91	0.83	0.87	290
14	0.93	0.90	0.92	301
15	0.98	0.93	0.95	278
16	0.86	0.85	0.85	274
17	0.89	0.90	0.89	370
18	0.71	0.76	0.73	295
19	0.66	0.65	0.66	280
20	0.93	0.94	0.94	334
21	0.89	0.94	0.91	352
22	0.89	0.94	0.92	343
23	0.93	0.89	0.91	175
24	0.98	0.89	0.93	161
accuracy			0.86	7048
macro avg	0.86	0.85	0.86	7048
weighted avg	0.86	0.86	0.86	7048

LogisticRegression:

	precision	recall	f1-score	support
0	0.79	0.78	0.79	257
1	0.77	0.80	0.78	295
2	0.79	0.78	0.78	206
3	0.81	0.78	0.80	227
4	0.87	0.88	0.87	230
5	0.87	0.87	0.87	401
6	0.84	0.85	0.85	272
7	0.91	0.90	0.91	219
8	0.95	0.90	0.93	220
9	0.92	0.93	0.92	299
10	0.89	0.94	0.91	309
11	0.91	0.90	0.90	425
12	0.86	0.83	0.85	235
13	0.89	0.84	0.86	290
14	0.92	0.89	0.91	301
15	0.99	0.92	0.96	278
16	0.86	0.83	0.84	274
17	0.93	0.88	0.90	370
18	0.70	0.75	0.73	295
19	0.66	0.68	0.67	280
20	0.93	0.95	0.94	334
21	0.87	0.96	0.92	352
22	0.84	0.95	0.89	343
23	0.94	0.90	0.92	175
24	0.98	0.81	0.88	161
accuracy			0.86	7048
macro avg	0.87	0.86	0.86	7048
weighted avg	0.87	0.86	0.87	7048

With n-grams:
kNN:

	precision	recall	f1-score	support
0	0.35	0.59	0.44	295
1	0.23	0.56	0.33	268
2	0.40	0.52	0.45	209
3	0.44	0.43	0.43	256
4	0.34	0.36	0.35	269
5	0.52	0.53	0.53	295
6	0.42	0.45	0.43	281
7	0.54	0.50	0.52	220
8	0.56	0.56	0.56	241
9	0.52	0.52	0.52	241
10	0.74	0.54	0.62	283
11	0.82	0.49	0.62	262
12	0.44	0.41	0.42	222
13	0.56	0.49	0.52	311
14	0.70	0.53	0.60	280
15	0.91	0.89	0.90	257
16	0.61	0.42	0.50	304
17	0.65	0.65	0.65	429
18	0.61	0.51	0.55	517
19	0.65	0.36	0.46	344
20	0.81	0.54	0.65	373
21	0.81	0.58	0.67	271
22	0.40	0.55	0.46	284
23	0.68	0.46	0.55	201
24	0.23	0.45	0.31	135
accuracy			0.52	7048
macro avg	0.56	0.52	0.52	7048
weighted avg	0.57	0.52	0.53	7048

MultinomialNB:

	precision	recall	f1-score	support
0	0.73	0.64	0.68	295
1	0.78	0.67	0.72	268
2	0.84	0.66	0.74	209
3	0.63	0.71	0.67	256
4	0.71	0.65	0.68	269
5	0.88	0.69	0.78	295
6	0.73	0.78	0.75	281
7	0.84	0.85	0.85	220
8	0.83	0.83	0.83	241
9	0.93	0.85	0.89	241
10	0.98	0.80	0.88	283
11	0.95	0.79	0.86	262
12	0.73	0.78	0.76	222
13	0.95	0.65	0.77	311
14	0.93	0.75	0.83	280
15	0.94	0.89	0.91	257
16	0.76	0.63	0.69	304
17	0.91	0.70	0.79	429
18	0.54	0.74	0.62	517
19	0.67	0.54	0.60	344
20	0.87	0.79	0.83	373
21	0.96	0.77	0.85	271
22	0.35	0.98	0.51	284
23	0.67	0.88	0.76	201
24	0.98	0.65	0.78	135
accuracy			0.74	7048
macro avg	0.80	0.75	0.76	7048
weighted avg	0.79	0.74	0.75	7048

SGDClassifier:

	precision	recall	f1-score	support
0	0.72	0.76	0.74	295
1	0.77	0.82	0.79	268
2	0.77	0.78	0.78	209
3	0.72	0.77	0.74	256
4	0.81	0.71	0.76	269
5	0.85	0.83	0.84	295
6	0.81	0.85	0.83	281
7	0.92	0.83	0.87	220
8	0.93	0.88	0.90	241
9	0.92	0.94	0.93	241
10	0.97	0.96	0.97	283
11	0.94	0.90	0.92	262
12	0.82	0.78	0.80	222
13	0.87	0.88	0.87	311
14	0.88	0.93	0.90	280
15	0.99	0.97	0.98	257
16	0.84	0.78	0.81	304
17	0.87	0.91	0.89	429
18	0.81	0.82	0.82	517
19	0.67	0.69	0.68	344
20	0.94	0.94	0.94	373
21	0.93	0.92	0.93	271
22	0.89	0.92	0.90	284
23	0.95	0.91	0.93	201
24	0.92	0.90	0.91	135
accuracy			0.85	7048
macro avg	0.86	0.86	0.86	7048
weighted avg	0.86	0.85	0.85	7048

LogisticRegression:

	precision	recall	f1-score	support
0	0.75	0.77	0.76	295
1	0.77	0.81	0.79	268
2	0.82	0.79	0.80	209
3	0.71	0.77	0.74	256
4	0.82	0.75	0.79	269
5	0.84	0.83	0.84	295
6	0.81	0.85	0.83	281
7	0.90	0.86	0.88	220
8	0.97	0.88	0.92	241
9	0.93	0.93	0.93	241
10	0.92	0.94	0.93	283
11	0.94	0.90	0.92	262
12	0.80	0.81	0.81	222
13	0.91	0.87	0.89	311
14	0.95	0.93	0.94	280
15	0.98	0.97	0.98	257
16	0.85	0.77	0.81	304
17	0.88	0.91	0.89	429
18	0.78	0.85	0.81	517
19	0.70	0.68	0.69	344
20	0.93	0.95	0.94	373
21	0.93	0.93	0.93	271
22	0.90	0.94	0.92	284
23	0.95	0.92	0.93	201
24	0.92	0.88	0.90	135
accuracy			0.86	7048
macro avg	0.87	0.86	0.86	7048
weighted avg	0.86	0.86	0.86	7048

Doc2Vec:
Train – 80% and test – 20%
Accuracy between actual and predicted labels:

```
similarity_accuracy = accuracy_score(y_test, y_pred_similarity)
print(f"Accuracy based on similarity to mean vectors: {similarity_accuracy}")
```

```
Accuracy based on similarity to mean vectors: 0.6993944636678201
```

kNN with predicted labels:

```
knn_classifier121 = KNeighborsClassifier(n_neighbors=5)
knn_classifier121.fit(X_train_np, y_train)
y_knn_pred121= knn_classifier121.predict(X_test_np)
knn_accuracy121 = accuracy_score(y_pred_similarity, y_knn_pred121)
print(f"Accuracy using k-Nearest Neighbors: {knn_accuracy121}")
```

```
Accuracy using k-Nearest Neighbors: 0.4532871972318339
```

kNN with actual labels:

```
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train_np, y_train)
y_knn_pred = knn_classifier.predict(X_test_np)
knn_accuracy = accuracy_score(y_test, y_knn_pred)
print(f"Accuracy using k-Nearest Neighbors: {knn_accuracy}")
```

```
Accuracy using k-Nearest Neighbors: 0.4742647058823529
```

MultinomialNB with predicted labels:

```
nb_classifier121 = MultinomialNB()
nb_classifier121.fit(X_train_shifted, y_train)
y_nb_pred121 = nb_classifier121.predict(X_test_shifted)
nb_accuracy121 = accuracy_score(y_pred_similarity, y_nb_pred121)
print(f"Accuracy using MultinomialNB: {nb_accuracy121}")
```

```
Accuracy using MultinomialNB: 0.8979238754325259
```

SGDClassifier with predicted labels:

```
sgd_classifier121 = SGDClassifier()
sgd_classifier121.fit(X_train_np, y_train)
y_sgd_pred121 = sgd_classifier121.predict(X_test_np)
sgd_accuracy121 = accuracy_score(y_pred_similarity, y_sgd_pred121)
print(f"Accuracy using SGDClassifier: {sgd_accuracy121}")
```

```
Accuracy using SGDClassifier: 0.7004757785467128
```

SGDClassifier with actual labels:

```
sgd_classifier = SGDClassifier()
sgd_classifier.fit(X_train_np, y_train)
y_sgd_pred = sgd_classifier.predict(X_test_np)
sgd_accuracy = accuracy_score(y_test, y_sgd_pred)
print(f"Accuracy using SGDClassifier: {sgd_accuracy}")
```

```
Accuracy using SGDClassifier: 0.6468425605536332
```

LogisticRegression with predicted labels:

```
logistic_regression121 = LogisticRegression(max_iter=1000)
logistic_regression121.fit(X_train_np, y_train)
y_logistic_pred121 = logistic_regression.predict(X_test_np)
logistic_accuracy121 = accuracy_score(y_pred_similarity, y_logistic_pred121)
print(f"Accuracy using Logistic Regression: {logistic_accuracy121}")
```

```
Accuracy using Logistic Regression: 0.7850346020761245
```

LogisticRegression with actual labels:

```
logistic_regression = LogisticRegression(max_iter=1000)
logistic_regression.fit(X_train_np, y_train)
y_logistic_pred = logistic_regression.predict(X_test_np)
logistic_accuracy = accuracy_score(y_test, y_logistic_pred)
print(f"Accuracy using Logistic Regression: {logistic_accuracy}")
```

```
Accuracy using Logistic Regression: 0.721885813148789
```

With train – 30% and test – 70%

Accuracy between actual and predicted labels:

```
mean_accuracy1 = accuracy_score(y_test1, y_pred1)
print(f"Accuracy based on mean vectors: {mean_accuracy1}")
```

```
Accuracy based on mean vectors: 0.6968423654452203
```

kNN with predicted labels:

```
knn_classifier11 = KNeighborsClassifier(n_neighbors=5)
knn_classifier11.fit(X_train_np1, y_train1)
y_knn_pred11 = knn_classifier11.predict(X_test_np1)
knn_accuracy11 = accuracy_score(y_pred1, y_knn_pred1)
print(f"Accuracy using k-Nearest Neighbors: {knn_accuracy11}")
```

```
Accuracy using k-Nearest Neighbors: 0.3529011926095285
```

kNN with actual labels:

```
knn_classifier1 = KNeighborsClassifier(n_neighbors=5)
knn_classifier1.fit(X_train_np1, y_train1)
y_knn_pred1 = knn_classifier1.predict(X_test_np1)
knn_accuracy1 = accuracy_score(y_test1, y_knn_pred1)
print(f"Accuracy using k-Nearest Neighbors: {knn_accuracy1}")
```

```
Accuracy using k-Nearest Neighbors: 0.35302477908916763
```

MultinomialNB with predicted labels:

```
nb_classifier2 = MultinomialNB()
nb_classifier2.fit(X_train_shifted1, y_train1)
y_nb_pred2 = nb_classifier2.predict(X_test_shifted1)
nb_accuracy2 = accuracy_score(y_pred1, y_nb_pred2)
print(f"Accuracy using MultinomialNB: {nb_accuracy2}")
```

```
Accuracy using MultinomialNB: 0.9024284743249088
```

MultinomialNB with actual labels:

```
nb_classifier2 = MultinomialNB()
nb_classifier2.fit(X_train_shifted1, y_train1)
y_nb_pred2 = nb_classifier2.predict(X_test_shifted1)
nb_accuracy2 = accuracy_score(y_pred1, y_nb_pred2)
print(f"Accuracy using MultinomialNB: {nb_accuracy2}")
```

```
Accuracy using MultinomialNB: 0.9024284743249088
```

SGDClassifieer with predicted labels:

```
sgd_classifier12 = SGDClassifier()
sgd_classifier12.fit(X_train_np1, y_train1)
y_sgd_pred12 = sgd_classifier12.predict(X_test_np1)
sgd_accuracy12 = accuracy_score(y_pred1, y_sgd_pred12)
print(f"Accuracy using SGDClassifier: {sgd_accuracy12}")
```

```
Accuracy using SGDClassifier: 0.6938762899338813
```

SGDClassifier with actual labels:

```
sgd_classifier1 = SGDClassifier()
sgd_classifier1.fit(X_train_np1, y_train1)
y_sgd_pred11 = sgd_classifier1.predict(X_test_np1)
sgd_accuracy1 = accuracy_score(y_test1, y_sgd_pred11)
print(f"Accuracy using SGDClassifier: {sgd_accuracy1}")
```

```
Accuracy using SGDClassifier: 0.6150899091639375
```

LogisticRegression with predicted labels:

```
logistic_regression11 = LogisticRegression(max_iter=1000)
logistic_regression11.fit(X_train_np1, y_train1)
y_logistic_pred11 = logistic_regression.predict(X_test_np1)
logistic_accuracy11 = accuracy_score(y_pred1, y_logistic_pred1)
print(f"Accuracy using Logistic Regression: {logistic_accuracy1}")
```

```
Accuracy using Logistic Regression: 0.7766174380522771
```

LogisticRegression with actual labels:

```
logistic_regression1 = LogisticRegression(max_iter=1000)
logistic_regression1.fit(X_train_np1, y_train1)
y_logistic_pred1 = logistic_regression.predict(X_test_np1)
logistic_accuracy1 = accuracy_score(y_test1, y_logistic_pred1)
print(f"Accuracy using Logistic Regression: {logistic_accuracy1}")
```

```
Accuracy using Logistic Regression: 0.7766174380522771
```

With unlabeled dataset:

kNN:

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'KNeighborsClassifier Accuracy: {accuracy_knn:.2f}')
```

```
KNeighborsClassifier Accuracy: 0.66
```

MultinomialNB:

```
mnb = MultinomialNB()
mnb.fit(X_train_shifted, y_train)
y_pred_mnb = mnb.predict(X_test_shifted)
accuracy_mnb = accuracy_score(y_test, y_pred_mnb)
print(f'MultinomialNB Accuracy: {accuracy_mnb:.2f}')
```

```
MultinomialNB Accuracy: 0.37
```

SGDClassifier:

```
sgd = SGDClassifier()
sgd.fit(X_train, y_train)
y_pred_sgd = sgd.predict(X_test)
accuracy_sgd = accuracy_score(y_test, y_pred_sgd)
print(f'SGDClassifier Accuracy: {accuracy_sgd:.2f}')
```

```
SGDClassifier Accuracy: 0.60
```

LogisticRegression:

```
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print(f'LogisticRegression Accuracy: {accuracy_lr:.2f}')
```

```
LogisticRegression Accuracy: 0.71
```

5.2 With clustering (ongoing development):

Here, I performed clustering (k-means) on unlabeled dataset. And calculate means of the documents that belong to a specific class in labeled dataset and also the means of documents that belong to the specific cluster. Below is the result:

accuracy			0.84	7048
macro avg	0.87	0.83	0.84	7048
weighted avg	0.85	0.84	0.84	7048

In this too, 84% is the accuracy.

5.3 Discussion:

First, the algorithm is tried on a small dataset that is Five News Groups (BBC). After converting the text data to vectors or numerical representation, the dataset is split into train and test as 80% and 20%. Then, the means of the documents that belong to a specific class in the training set are calculated and calculate the similarity between the means and the vector representations of each document in the test set. In this too, the accuracy between the actual and assigned labels gives above 95%, means the algorithm works well.

Then, again the dataset is split into train and test as 30% and 70%, applied the same algorithm. Also the accuracy between the actual and assigned labels stills gives over 95%. Then, four classifiers (with actual labels and assigned labels) are trained which give good accuracy. If the similarity finds maximum between the mean and the document, then the class label of the document is the class label of that class that the mean is used in finding the similarity.

Second, the algorithm is applied to the larger dataset which has 25 classes. All the procedures are same as explained above, and the accuracy between actual and assigned labels gives over 80%, means the algorithm works well. However, the feature extraction technique which gives such accuracy is both TFIDF and TFIDF with n-grams. If the similarity finds maximum between the mean and the document, then the class label of the document is the class label of that class that the mean is used in finding the similarity.

Then, with both labeled and unlabeled datasets, the same algorithm is applied. In this, we want to say something. In gathering raw text data gives us a long time to label all the data. What if we can label only some data with our required class labels and then assign or predict the labels of the remaining text data? Now, the proposed algorithm is here to accomplish this.

The means of the documents of the labeled dataset are calculated, and then find similarity between the means and the documents in the unlabeled dataset. If the similarity finds maximum between the mean and the document, then the class label of the document is the class label of that class that the mean is used in finding the similarity. So, maximum the similarity is the class labels of the documents in the unlabeled dataset.

Four classifiers are trained with feature extraction techniques CountVectorizer, CountVectorizer with n-grams, TFIDF, TFIDF with n-grams and Doc2Vec. Below are the results:

CountVectorizer	**Train : Test = 80 : 20 With predicted labels**	**Train : Test = 80 : 20 With actual labels**	**Train : Test = 30 : 70 With predicted labels**	**Train : Test = 30 : 70 With actual labels**
kNN	43%	52%	31%	36%
MultinomialNB	69%	87%	67%	82%
SGDClassifier	65%	84%	65%	78%
LogisticRegression	67%	87%	68%	83%

Table-1: CountVectorizer

TFIDF	**Train : Test = 80 : 20 With predicted labels**	**Train : Test = 80 : 20 With actual labels**	**Train : Test = 30 : 70 With predicted labels**	**Train : Test = 30 : 70 With actual labels**
kNN	78%	82%	77%	76%
MultinomialNB	85%	86%	86%	82%
SGDClassifier	87%	89%	88%	87%
LogisticRegression	90%	88%	92%	85%

Table-2: TFIDF

CountVectorizer n-grams	Train : Test = 80 : 20 With predicted labels	Train : Test = 80 : 20 With actual labels	Train : Test = 30 : 70 With predicted labels	Train : Test = 30 : 70 With actual labels
kNN	43%	51%	36%	39%
MultinomialNB	71%	84%	71%	82%
SGDClassifier	65%	84%	64%	77%
LogisticRegression	67%	86%	67%	82%

Table-3: CountVectorizer with n-grams

TFIDF n-grams	Train : Test = 80 : 20 With predicted labels	Train : Test = 80 : 20 With actual labels	Train : Test = 30 : 70 With predicted labels	Train : Test = 30 : 70 With actual labels
kNN	75%	78%	72%	72%
MultinomialNB	87%	84%	87%	82%
SGDClassifier	86%	88%	85%	86%
LogisticRegression	89%	87%	91%	85%

Table-4: TFIDF with n-grams

CountVectorizer with unlabeled dataset:

CountVectorizer	Accuracy	With n-grams
kNN	50%	52%
MultinomialNB	74%	74%
SGDClassifier	86%	85%
LogisticRegression	86%	86%

Table-5: CountVectorizer with unlabeled dataset

TFIDF with unlabeled

TFIDF	Accuracy	With n-grams
kNN	70%	68%
MultinomialNB	81%	78%
SGDClassifier	88%	87%
LogisticRegression	86%	85%

Table-6: TFIDF with unlabeled dataset

Doc2Vec:

Doc2Vec	Train : Test = 80 : 20 With predicted labels	Train : Test = 80 : 20 With actual labels	Train : Test = 30 : 70 With predicted labels	Train : Test = 30 : 70 With actual labels
kNN	45%	47%	35%	35%
MultinomialNB	89%	66%	90%	66%
SGDClassifier	70%	64%	69%	61%
LogisticRegression	78%	72%	77%	77%

Table-7: Doc2Vec

Doc2Vec with unlabeled dataset:

Doc2Vec	Accuracy
kNN	66%
MultinomialNB	37%
SGDClassifier	60%
LogisticRegression	71%

Table-8: Doc2Vec with unlabeled dataset

Chapter 6

CONCLUSION

Sometimes we face problems to label the raw text data while we are gathering them. There is semi-supervised learning in traditional ML types. But why don't we try another approach? So in this project, we present the similarity measure. To label the raw text data with the help of related labeled data, we can assign or predict or give the labels to those unlabeled raw text data.

In this project, ML techniques to classify the text (categories) and ten steps in preprocessing are used, the most important steps are standardization, tokenization, removing stop words and lemmatization (feature engineering). The dataset we got is raw dataset, no preprocessed means the unwanted things or noises are in the dataset, so preprocessing is the way to remove these.

The text data cannot be understood by classifiers, classifiers only understand the numerical values or data. So, we need to convert the preprocessed text data into numerical data. In this, CountVectorizer, CountVectorizer with n-grams, TFIDF, TFIDF with n-grams and Doc2Vec are used. These are feature extraction techniques.

After converting the text data to respective numerical forms or vectors, we can find the means (mean vectors) of the documents that belong to a specific class in labeled dataset, using these means and vector representations of the unlabeled documents we can find similarity between them. Then we can assign the class labels of the labeled dataset to the document in the unlabeled dataset where the similarity is the maximum.

First, using labeled dataset, and after splitting into train and test. The test set without the actual labels, calculate the similarity to assign the class labels (or we can say predicted labels). Then, using both labeled and unlabeled datasets, we can do the same.

6.1 Future Scope:

Human languages are difficult to understand, even by us. We need to make the machines understand human languages to ease our daily life. This is where, NLP comes into play.

There are many raw text data that we do not know what class or category or group do they belong. So, we can give the labels using this project, the similarity measure. As mentioned above, we can use any text vectorization or text representation to achieve this and we can assign class labels to the unlabeled documents using the similarity measure. In this project, I use cosine similarity.

We did this project with ML techniques, but we can also do this using DL techniques. We didn't do the data augmentation here in my project, we can also do this. In NLP, we have easy data augmentation, backtranslation and generative models to perform data augmentation. Using these, may increase the accuracy.

First, we used labeled dataset with 25 classes. One can also use a labeled dataset with many classes. Then, We used both labeled and unlabeled datasets.

CountVectorizer, CountVectorizer with n-grams, TFIDF, TFIDF with n-grams and Doc2Vec are the feature extraction techniques I used in this project. We can also use Word2Vec, FastText, Glove in feature extraction. This may rise the accuracy and predict with higher accuracy too. I didn't use dimensional reduction techniques like PCA, LDA, etc. we can also use this to reduce the dimension. Dimensional reduction techniques are the ones that can reduce the high dimension to lower dimension ones. Means, a large number of features can be reduced to a lower number of features. In the extraction with the vectorizers, using n-grams features gives us a large vector space, so we can use dimensional reduction techniques.

The classifiers I used are from ML. I also think, we can use CNN (Convolution Neural Networks), RNN (Recurrent Neural Networks), Transformer, LSTM, etc. These are DL models.

REFERENCES

[1] Sarkar D. 2016. “Text Analytics with Python- A practical Real-World Approach to Gaining Actionable Insights from Your Data”. Apress.

[2] Vivian Lay Shan Lee, Keng Hoon Gan, Tien Ping Tan, Rosni Abdullah. “Semi-supervised Learning for Sentiment Classification using Small Number of Labeled Data. (https://www.sciencedirect.com/science/article/pii/S1877050919318708)

[3] idera.com/glossary/data-growth/

[4] Phiri M. 2022. “Exponential Growth of Data”. medium.com/@mwalimu/exponentialgrowth-of-data-2f53df89124

[5] Afzal A. 2022. “Step-by-step Explanation of Text Classification”. (https://www.analyticsvidhya.com/blog/2022/08/step-by-step-explanation-of-text-classification/)

[6] Akshay Sharma. Brief Introduction to N-gram and TF-IDF | Tokenization.

(https://medium.com/in-pursuit-of-artificial-intelligence/brief-introduction-to-n-gram-and-tf-idf-tokenization-e58d22555bab)

[7] Mwalimu Phiri, 2022. (https://medium.com/@mwaliph/exponential-growth-of-data-2f53df89124)

[8] Shivam5992 Bansal, 2024. (https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/)

[10] Scikit-learn.org

[11] Igor Assis Braga, Maria Carolina Monard, Edson Takashi Matsubara. “Combining nigrams and Bigrams in Semi-Supervised Text Classification”. (https://www.researchgate.net/profile/Maria-Carolina-Monard/publication/228678357_Combining_unigrams_and_bigrams_in_semi-supervised_text_classification/links/544e6ac30cf2bca5ce90b302/Combining-unigrams-and-bigrams-in-semi-supervised-text-classification.pdf)

[12] Fabio Rangel, et. al. “Semi-Supervised Classification of Social Textual Data Using WiSARD”.
(https://www.researchgate.net/profile/Jonice-Oliveira/publication/319552892_Semi-Supervised_Classification_of_Social_Textual_Data_Using_WiSARD/links/5bc5e211458515f7d9bf6350/Semi-Supervised-Classification-of-Social-Textual-Data-Using-WiSARD.pdf)

[13] Deepshikha Kalita. “Supervised and Unsupervised Document Classification-A survey”.
(https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=bb6fb8b0a753d2bbec7b7ec25ff91e15548bd3bc)

[14] Kamal Nigam, et. al. “Text Classification from Labeled and Unlabeled Documents using EM”. (https://link.springer.com/article/10.1023/A:1007692713085)

[15] Youngjoong Ko, Jungyun Seo. “Automatic Text Categorization by Unsupervised Learning”. (https://aclanthology.org/C00-1066.pdf)

[16] Parlad Neupane, 2023.
(https://www.analyticsvidhya.com/blog/2020/12/understanding-text-classification-in-nlp-with-movie-review-example-example/)

[17] Sahel Eskandar, 2023. (https://medium.com/@eskandar.sahel/exploring-feature-extraction-techniques-for-natural-language-processing-46052ee6514)

[18] Shivam5992 Bansal. A Comprehensive Guide to Understand and Implement Text Classification in Python. (https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/)