Getting familiar with the dataset

In [ ]:

```
import pandas as pd

file_path = 'Project1-ClassificationDataset.csv'

#selected_columns = ["full_text", "summary", "keywords", "publish_date", "authors", "url", "leaf_label", "root_label"]

df = pd.read_csv(file_path)

num_rows, num_columns = df.shape

print(f"Number of rows (samples): {num_rows}")
print(f"Number of columns (features): {num_columns}")
```

```
Number of rows (samples): 3476
Number of columns (features): 8
```

Question 1:

1. Overview: Number of rows (samples): 3476, Number of columns (features): 8

In [ ]:

```
df.head()
```

Out[ ]:

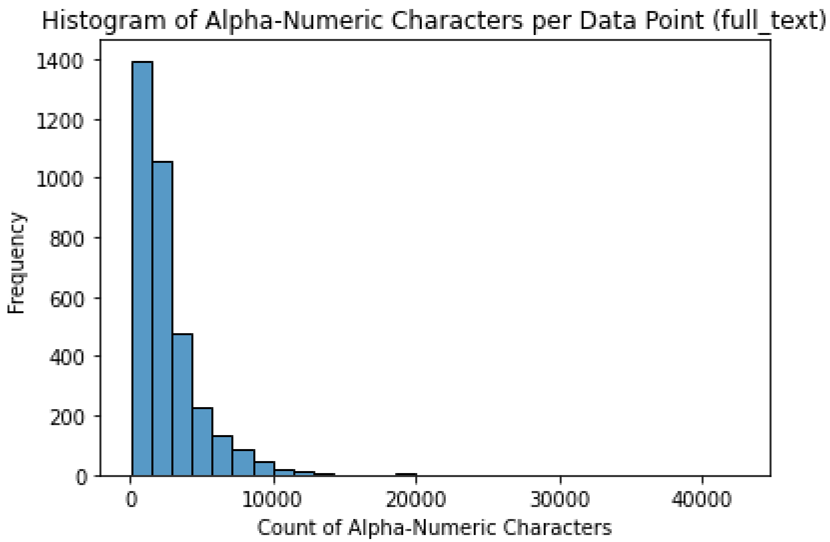
	full_text	summary	keywords	publish_date	authors	url	leaf_label	root_label
0	'Personalize Your NBA App Experience for the '...	'Personalize Your NBA App Experience for the '...	['original', 'content', 'live', 'slate', 'game...	NaN	['Official Release']	https://www.nba.com/news/nba-app-new-features-...	basketball	sports
1	'Mike Will attends the Pre-GRAMMY Gala and GRA...	'Mike WiLL Made-It has secured a partnership w...	['lead', 'espn', 'nbas', 'madeit', 'nba', 'lat...	2023-10-18 16:22:29+00:00	['Marc Griffin']	https://www.vibe.com/news/entertainment/mike-w...	basketball	sports

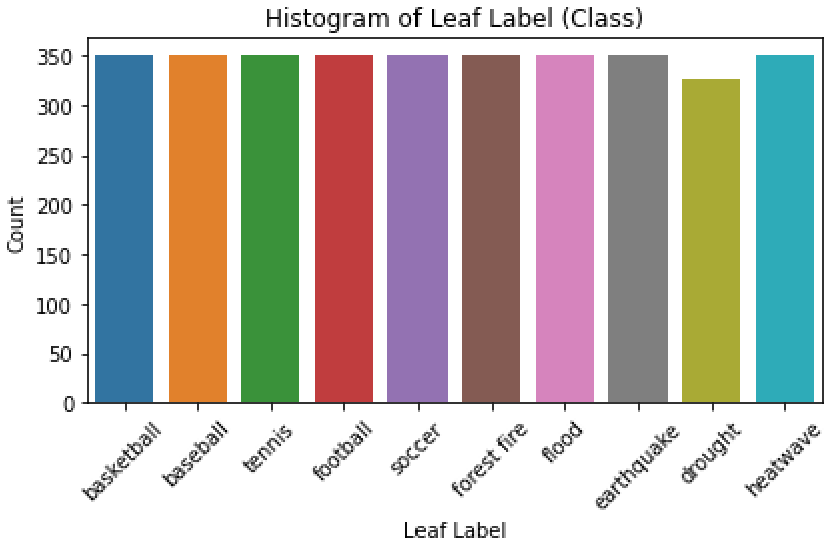
	full_text	summary	keywords	publish_date	authors	url	leaf_label	root_label
2	'The Golden State Warriors are struggling to f...	'The Golden State Warriors are struggling to f...	['insider', 'york', 'thing', 'nbc', 'tag', 'nb...	NaN	[]	https://www.nbcnewyork.com/tag/featured-nba/	basketball	sports
3	'On Nov. 28, the NBA and Nike will collaborate...	'On Nov. 28, the NBA and Nike will collaborate...	['watch', 'telecast', 'ultimate', 'membership'...	NaN	['Official Release']	https://www.nba.com/news/watch-nba-games-ultim...	basketball	sports
4	'The NBA announced additions and innovations t...	'The NBA announced additions and innovations t...	['experience', 'bring', 'media', 'crennan', 'n...	2023-10-17 12:00:17+00:00	['Chris Novak', 'About Chris Novak']	https://awfulannouncing.com/tech/nba-app-2023-...	basketball	sports

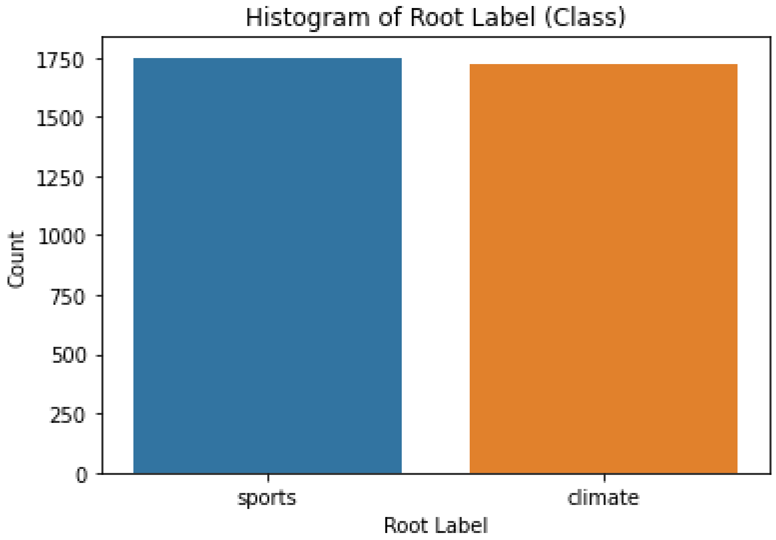
In [ ]:

```python
import matplotlib.pyplot as plt
import seaborn as sns

# (a) Histogram for the total number of alpha-numeric characters per data point in the feature 'full_text'
df['alpha_numeric_count'] = df['full_text'].apply(lambda x: sum(c.isalnum() for c in x))
#plt.figure(figsize=(10, 6))
sns.histplot(df['alpha_numeric_count'], bins=30, kde=False)
plt.title('Histogram of Alpha-Numeric Characters per Data Point (full_text)')
plt.xlabel('Count of Alpha-Numeric Characters')
plt.ylabel('Frequency')
plt.show()

# (b) Histogram for the column 'leaf_label' (class)
#plt.figure(figsize=(10, 6))
sns.countplot(x='leaf_label', data=df)
plt.title('Histogram of Leaf Label (Class)')
plt.xlabel('Leaf Label')
plt.xticks(rotation = 45)
plt.ylabel('Count')
plt.tight_layout()
plt.show()

# (c) Histogram for the column 'root_label' (class)
#plt.figure(figsize=(10, 6))
sns.countplot(x='root_label', data=df)
plt.title('Histogram of Root Label (Class)')
```

```python
plt.xlabel('Root Label')
plt.ylabel('Count')
plt.show()
```







2. Histograms:

(1). The first histograms shows the frequency of the length of text in the full-text column. The peak on the histogram is in the very left, indicates that this dataset mostly contain texts with very short lengths. As the text length increase, the frequency decrease dramatically, and becomes very small when text length reach to 10,000. This trends suggests that shorter texts are more common in this dataset.

(2). The second histograms shows the frequency of each class in the "leaf-label" column. This dataset have 10 leaf labels, and the frequency of each label is almost similar, approximately 350 datapoints. The exception is the drought label, which has slightly fewer dataset than the other categories.

(3). From the third histograms, this dataset only contains two root labels: sports and climate. Each root label have approximately 1750 datapoints.

Binary Classification

1. Splitting the entire dataset into training and testing data

In [ ]:

```
import numpy as np
import random
np.random.seed(42)
random.seed(42)
```

In [ ]:

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df[["full_text", "root_label"]], test_size=0.2, random_state=42)

print(f"Number of Training Samples: {train.shape[0]}")
print(f"Number of Testing Samples: {test.shape[0]}")
print(f"Training Samples Shape: {train.shape}")
print(f"Test Samples Shape: {test.shape}")
```

```
Number of Training Samples: 2780
Number of Testing Samples: 696
Training Samples Shape: (2780, 2)
Test Samples Shape: (696, 2)
```

Question 2:

- Number of Training Samples: 2780
- Number of Testing Samples: 696

2. Feature extraction

In [ ]:

```
import re
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from string import punctuation
from sklearn.feature_extraction import text

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

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Michael_Mi\AppData\Roaming\nltk_data...
```

```
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Michael_Mi\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Michael_Mi\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Michael_Mi\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[ ]: True

In [ ]:

```
import re
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from string import punctuation
from sklearn.feature_extraction import text

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

def clean(text):
    text = re.sub(r'^https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
    texter = re.sub(r"<br />", " ", text)
    texter = re.sub(r"&quot;", "\"", texter)
    texter = re.sub('&#39;', "\"", texter)
    texter = re.sub('\n', " ", texter)
    texter = re.sub(' u '," you ", texter)
    texter = re.sub('`',"", texter)
    texter = re.sub(' +', ' ', texter)
    texter = re.sub(r"(!)\1+", r"!", texter)
    texter = re.sub(r"(\?)\1+", r"?", texter)
    texter = re.sub('&amp;', 'and', texter)
```

```python
    texter = re.sub('\r', ' ',texter)
    clean = re.compile('<.*?>')
    texter = texter.encode('ascii', 'ignore').decode('ascii')
    texter = re.sub(clean, '', texter)
    if texter == "":
        texter = ""
    return texter

def is_number(s):
    return re.match(r'-?\d+\.?\d*', s) is not None

def penn2morphy(penntag):
    """ Converts Penn Treebank tags to WordNet. """
    morphy_tag = {'NN':'n', 'JJ':'a',
                  'VB':'v', 'RB':'r'}
    try:
        return morphy_tag[penntag[:2]]
    except:
        return 'n'

def lemmatize_sent(list_word):
    # Text input is string, returns array of lowercased strings(words).
    return [wnl.lemmatize(word.lower(), pos=penn2morphy(tag))
            for word, tag in pos_tag(list_word)]

def stem_rmv_punc(doc): # this should have been at the sentence-level because the pos-tag performs best at sentence-level
    return (word for word in lemmatize_sent(analyzer(doc)) if word not in combined_stopwords and not is_number(word))

train['cleaned_text'] = train['full_text'].apply(clean)
test['cleaned_text'] = test['full_text'].apply(clean)

stop_words_skt = text.ENGLISH_STOP_WORDS
stop_words_en = stopwords.words('english')
combined_stopwords = set.union(set(stop_words_en),set(punctuation),set(stop_words_skt))

wnl = nltk.wordnet.WordNetLemmatizer()
analyzer = CountVectorizer().build_analyzer()

count_vectorizer = CountVectorizer(min_df=3, stop_words='english', analyzer=stem_rmv_punc)
tfidf = TfidfTransformer()

X_train_count = count_vectorizer.fit_transform(train['cleaned_text'])
X_test_count = count_vectorizer.transform(test['cleaned_text'])
```

```
X_train_tfidf = tfidf.fit_transform(X_train_count)
X_test_tfidf = tfidf.transform(X_test_count)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Michael_Mi\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Michael_Mi\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Michael_Mi\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Michael_Mi\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\Michael_Mi\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [ ]:

```
print(f"Shape of TF-IDF-processed train matrix: {X_train_tfidf.shape}")
print(f"Shape of TF-IDF-processed test matrix: {X_test_tfidf.shape}")
```

```
Shape of TF-IDF-processed train matrix: (2780, 13173)
Shape of TF-IDF-processed test matrix: (696, 13173)
```

Question 3:

answer

3a.

3a.1 Lemmatization:

Pros:

```
  a. The root word is an actual word
  b. Using a dictionary to analysis word and can convery word into root form, have high accuracy and can maintain
the original context
```

cons:

```
a. Requires a detailed large dictionary to convert word
b. Needs more computational resource
c. Requires the word's correct part of speech to accurately lemmatize it
```

Effect on size: Lemmatization decreases the size of dictionary by convert words into root form

3a.2 Stemming

Pros:

```
a. Simple implementation and running faster than lemmatization
b. Only removes the end portions of a word to try to find its stem
```

cons:

```
a. Do not consider the context and can convert to wrong word, so the accuracy is low
```

Effect on size: Stemming decreases the size of dictionary, but might not decrease as much as Lemmatization.

3b.

Effect of min df on TF-IDF Matrix: A higher min_df means fewer words are included, filtering out some less common words, but some important subtle information may be lost. A lower min_df means that words that appear less often are allowed to be added, which is beneficial to capturing some uncommon subtle information, but it may contain noise, thus affecting the performance of the model.In short, the choice of min_df is very important

3c. Text Preprocessing Order:

The numbers and punctuations should be removed before Lemmatizing, since they are not contribute to the dictionary.

The stopwords should be removed after Lemmatizing, since Lemmatizer requires full sentence as input, including stopwords.

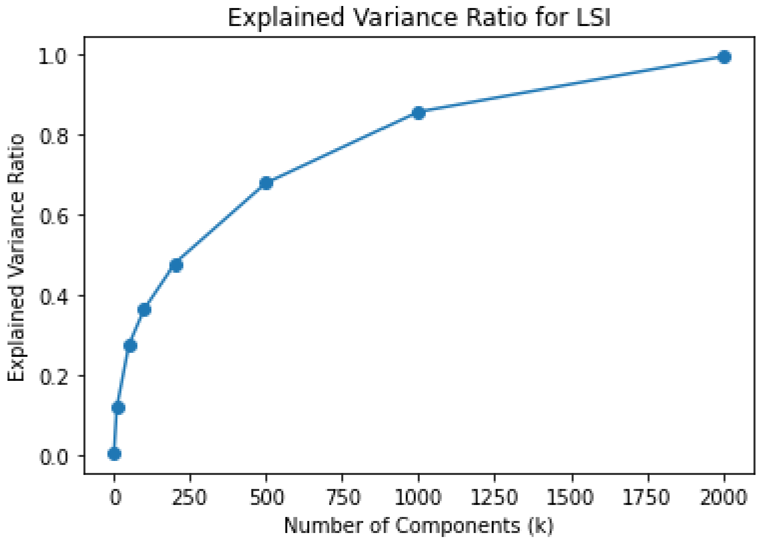
3d. TF-IDF Matrices Shape: The shape of the train matrix is (2780, 13173) and the shape of the test matrix is (696, 13173); both have the same number of rows as the result of Question 2.

3. Dimensionality Reduction

- 4a

In [ ]:

```
from sklearn.decomposition import TruncatedSVD
import matplotlib.pyplot as plt

k_values = [1, 10, 50, 100, 200, 500, 1000, 2000]

explained_variances = []

# for k in k_values:
#     svd = TruncatedSVD(n_components=k)
#     svd.fit(X_train_tfidf)  # X_train_tfidf
#     explained_variances.append(svd.explained_variance_ratio_.sum())
svd_list = []
for i in range(len(k_values)):
    # svd = TruncatedSVD(n_components=k)
    svd_list.append(TruncatedSVD(n_components=k_values[i]))
    svd_list[i].fit(X_train_tfidf)  # X_train_tfidf
    explained_variances.append(svd_list[i].explained_variance_ratio_.sum())

plt.plot(k_values, explained_variances, marker='o')
plt.xlabel('Number of Components (k)')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio for LSI')
plt.show()
```



Questation 4

4a-answer.

Explained Variance Ratio Plot for LSI

The var ratio plot for LSI indicates the extend of how the total var in TF_IDF is retained as increasing number of k, and overall, as k increases, the explained var ratio also increases, which means that more information is captured.

Concavity in the context of dimensionality reduction implies a trade-off between the retained elements and the explained variance. Increasing the value of K initially leads to a significant rise in explained variance; however, as K further increases, the growth in explained variance gradually plateaus. Hence, while a larger K corresponds to a greater explained variance, the incremental gain diminishes with higher K values. Moreover, higher K values result in prolonged algorithm runtime. Enhanced explained variance signifies a preservation of more data as K increases, yet in the case of dimensionality reduction for categorized datasets, retaining more data is crucial for maintaining higher test accuracy.

- 4b

In [ ]:

```
from sklearn.decomposition import NMF
from numpy.linalg import norm
from sklearn.metrics import mean_squared_error
```

```python
k = 50

lsi_model = TruncatedSVD(n_components=k)
train_lsi = lsi_model.inverse_transform(lsi_model.fit_transform(X_train_tfidf))
# lsi_mse = norm(np.subtract(X_train_tfidf.toarray(), train_lsi), 'fro')**2
lsi_mse = mean_squared_error(X_train_tfidf.toarray(), train_lsi)

nmf_model = NMF(n_components=k, init='random', max_iter=300)
W = nmf_model.fit_transform(X_train_tfidf)
H = nmf_model.components_
train_nmf = nmf_model.inverse_transform(W)
# nmf_mse = norm(np.subtract(X_train_tfidf.toarray(), train_nmf), 'fro')**2
nmf_mse = mean_squared_error(X_train_tfidf.toarray(), train_nmf)

print('LSI MSE:', lsi_mse)
print('NMF MSE:', nmf_mse)
```

```
LSI MSE: 5.317909141726312e-05
NMF MSE: 5.409425986967508e-05
```

4b-answer.

The NMF results in a higher residual MSE error, with ‖X−WH‖2F = 5.409425986967508e-05, compared to ‖X−UkΣkVTk‖2F = 5.317909141726312e-05. This increased error is attributed to the constraints imposed by NMF on matrices W and H, limiting them to non-negative values. In contrast, LSI does not enforce such restrictions. The truncated SVD used by LSI incorporates negative elements, providing a less constrained approximation of X. This allows for a more flexible representation in reduced dimensionality, contributing to the contrast in error rates between NMF and LSI.

4. Classification Algorithms

Question 5

- 5a

In [ ]:

```python
from sklearn import svm
from sklearn import metrics
import seaborn as sn
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import TruncatedSVD
```

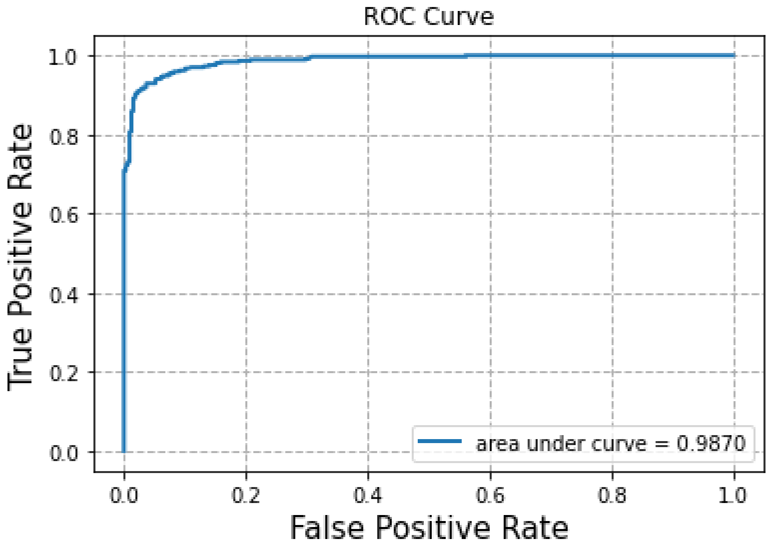
```python
# Apply LSI on data
svd_components = 50
truncated_svd = TruncatedSVD(n_components=svd_components, random_state=42)
train_truncated = truncated_svd.fit_transform(X_train_tfidf)
test_truncated = truncated_svd.transform(X_test_tfidf)
```

In [ ]:

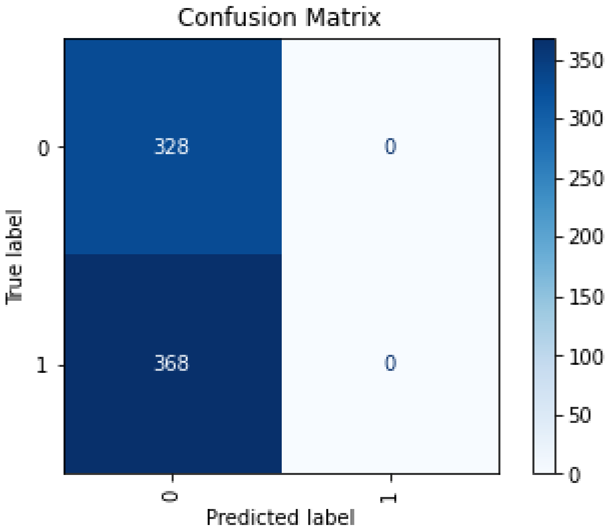
```python
from sklearn.svm import SVC
import matplotlib.pyplot as plt

def train_evaluate_SVM(train_data, train_labels, test_data, test_labels, C_value):
    # Train SVM
    svm_model = SVC(kernel='linear', C=C_value, random_state=42)
    svm_model.fit(train_data, train_labels)

    # Plot ROC curve
    prob_estimates = svm_model.decision_function(test_data)
    fpr, tpr, _ = metrics.roc_curve(test_labels, prob_estimates)
    roc_auc = metrics.auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label='area under curve = %0.4f' % roc_auc)
    plt.grid(color='0.7', linestyle='--', linewidth=1)
    plt.title('ROC Curve')
    plt.xlabel('False Positive Rate', fontsize=15)
    plt.ylabel('True Positive Rate', fontsize=15)
    plt.legend(loc="lower right")
    plt.show()

    # Print SVM metrics
    print(f"C = {C_value}")
    predictions = svm_model.predict(test_data)
    print('Accuracy:', metrics.accuracy_score(test_labels, predictions))
    print('Recall:', metrics.recall_score(test_labels, predictions))
    print('Precision:', metrics.precision_score(test_labels, predictions, zero_division=1))
    print('F-1 Score:', metrics.f1_score(test_labels, predictions))

    # Plot confusion matrix
    confusion_matrix = metrics.confusion_matrix(test_labels, predictions)
    confusion_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=np.unique(test_labels))
    confusion_display.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
    plt.title('Confusion Matrix')
```

In [ ]:

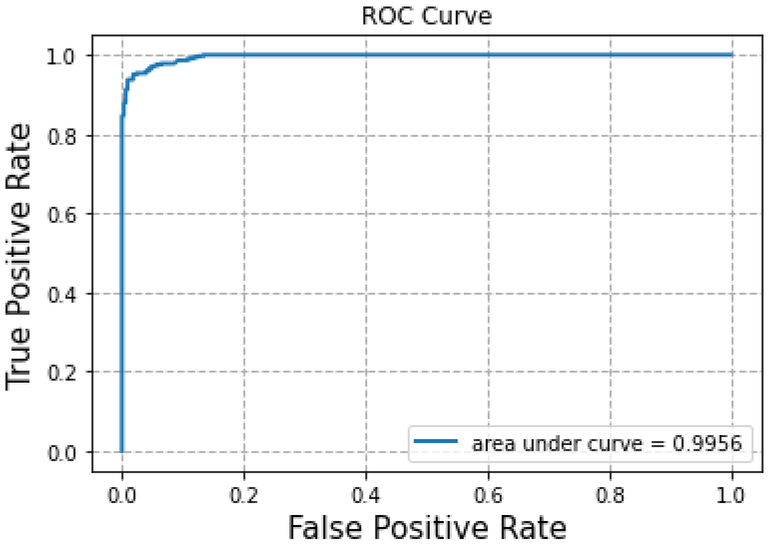
```
# Transform categories into numbers (Climate=0, Sports=1)
label_encoder = LabelEncoder()
train['binary_root_label'] = label_encoder.fit_transform(train['root_label'])
test['binary_root_label'] = label_encoder.fit_transform(test['root_label'])
categories = ['climate', 'sports']

train_evaluate_SVM(train_truncated, train['binary_root_label'], test_truncated, test['binary_root_label'], 0.0001)
```



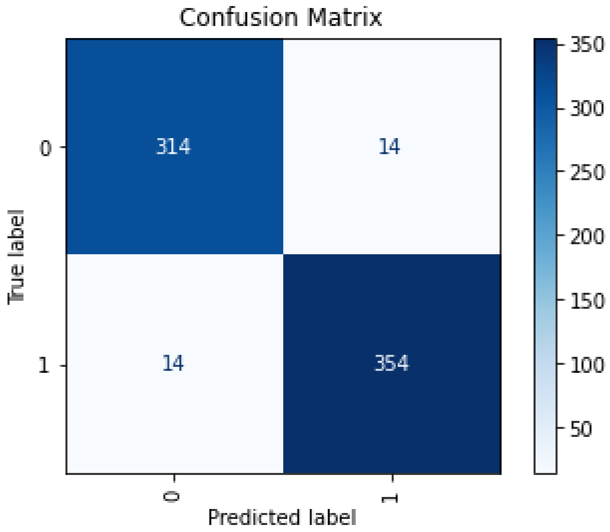

```
C = 0.0001
Accuracy: 0.47126436781609193
Recall: 0.0
Precision: 1.0
F-1 Score: 0.0
```



In [ ]:
```
train_evaluate_SVM(train_truncated, train['binary_root_label'], test_truncated, test['binary_root_label'], 1000)
```

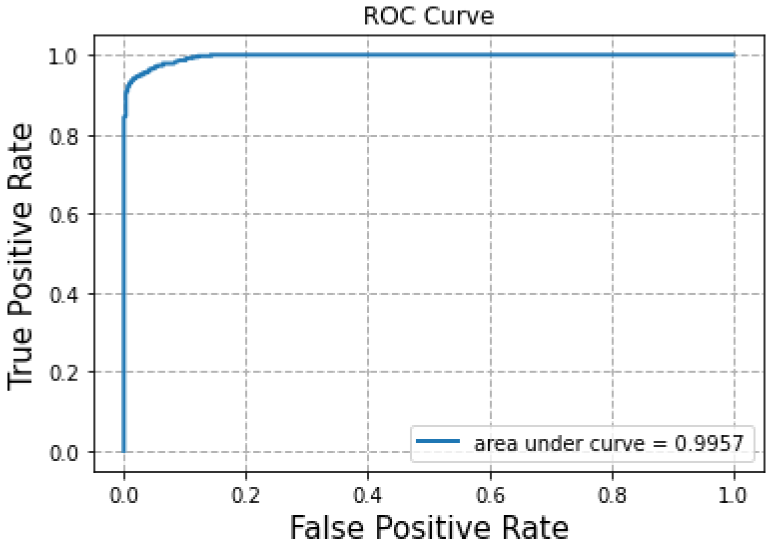



```
C = 1000
Accuracy: 0.9597701149425287
Recall: 0.9619565217391305
Precision: 0.9619565217391305
F-1 Score: 0.9619565217391305
```

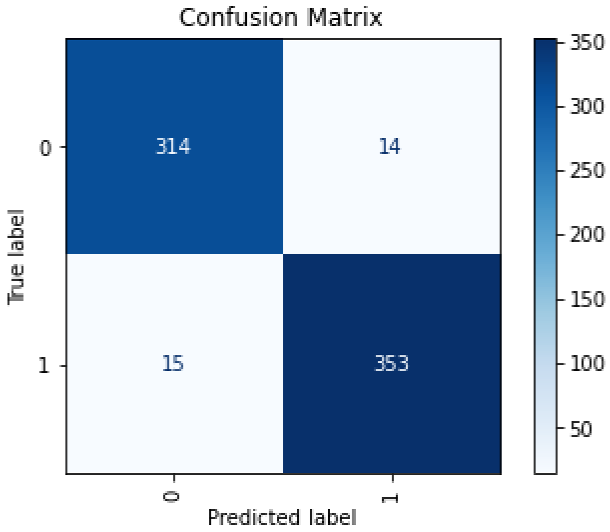


In [ ]:

```
train_evaluate_SVM(train_truncated, train['binary_root_label'], test_truncated, test['binary_root_label'], 100000)
```




```
C = 100000
Accuracy: 0.9583333333333334
Recall: 0.9592391304347826
Precision: 0.9618528610354223
F-1 Score: 0.9605442176870749
```



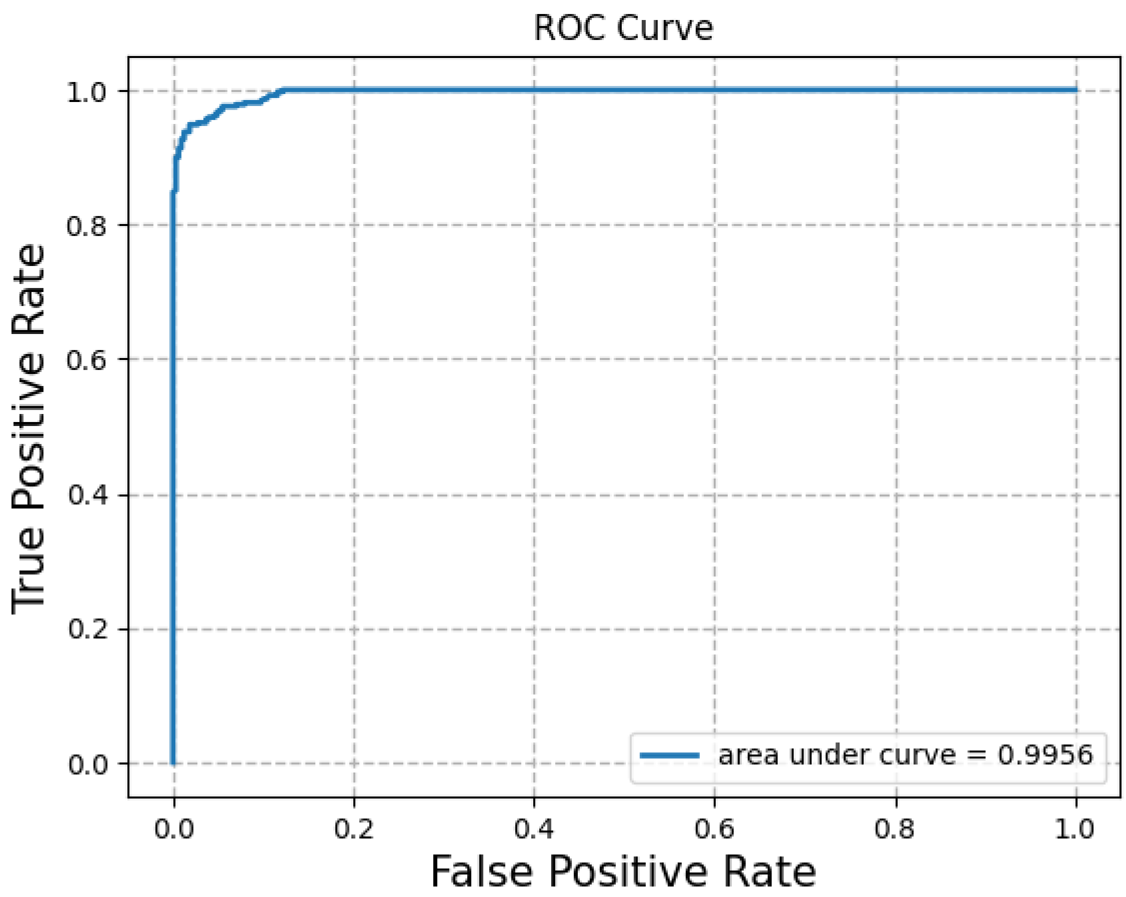
5a-answer:

- The SVM with a γ value of 1000 (hard margin) showcased superior performance, demonstrating higher accuracy, F-1 score, precision, and ROC area under the curve in comparison to the soft margin SVM. Also when The SVM with a γ value of 100000,the performance this similiar to γ=1000,a little bit worse than γ=1000.
- Gamma acts as a hyperparameter that governs SVM error during the training process. A low γ value, like 0.0001, leads to minimal errors in class separation, resulting in overfitting to the training data. This overfitting translates to low accuracy when validating on testing data, as indicated by the confusion matrix, which exposes misclassifications. The model accurately identified sports articles but inaccurately categorized all climate articles as sports, revealing a flawed classification.This also shows that when gamma is too small, the model cannot well distinguish the differences between the two categories in the two-classification task. It will be easier to summarize one category into another, and it will be more difficult to identify the two categories with different feature.
- Although the ROC curve for the soft margin SVM implies high separability, this metric alone does not necessarily align with accurate classification. A model with strong separability may still exhibit poor performance in terms of classification accuracy, underscoring the importance of considering multiple metrics for a thorough model evaluation.
- 5b

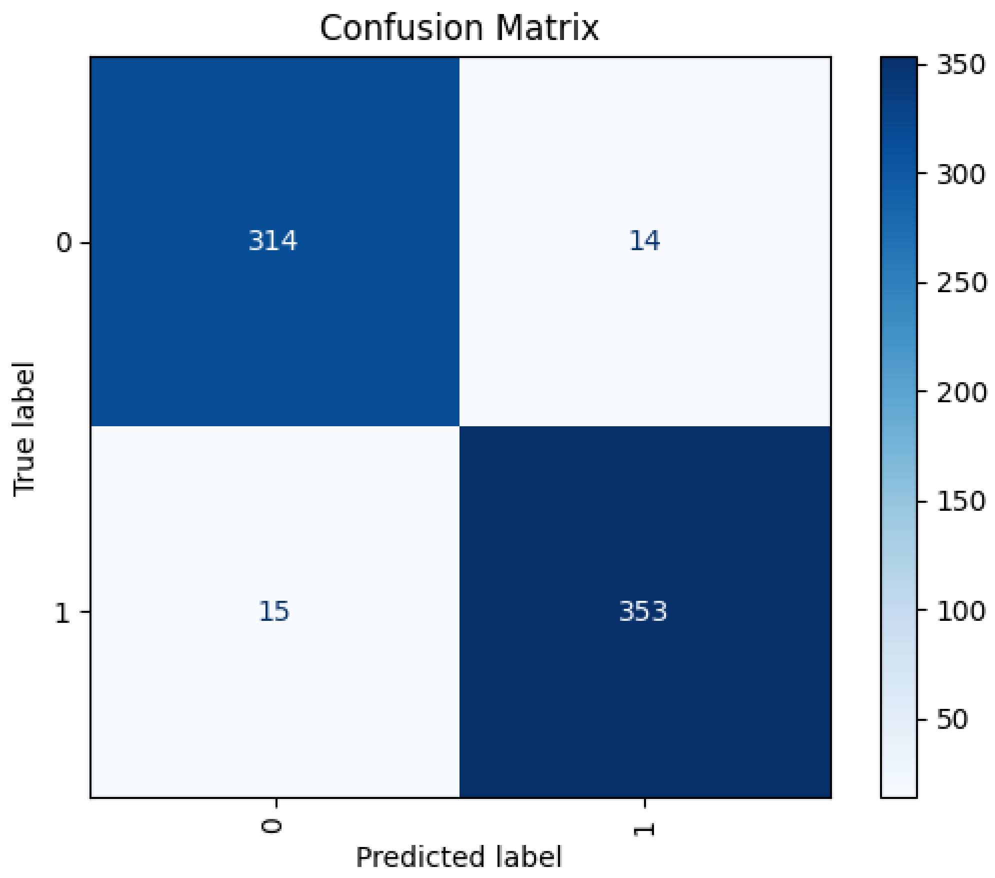
In [ ]:

```
from sklearn.model_selection import GridSearchCV

# Optimize SVM parameters with a linear kernel
c_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000]
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000],
              'kernel': ['linear']}
svm_model = SVC(random_state=42)
grid_search_cv = GridSearchCV(svm_model, param_grid, cv=5, refit=True, scoring='accuracy', n_jobs=-1)
grid_search_cv.fit(train_truncated, train['binary_root_label'])
predicted_labels = grid_search_cv.predict(test_truncated)

# Display cross-validation accuracy scores for different C values

for c in c_values:
    print(f'C Parameter: {c}, Cross-Validation Accuracy: {grid_search_cv.cv_results_["mean_test_score"][c_values.index(c)]}')

# Output the best C parameter
print("\nOptimal C Parameter: ", grid_search_cv.best_params_['C'])

# Evaluate SVM performance using the best C parameter
train_evaluate_SVM(train_truncated, train['binary_root_label'], test_truncated, test['binary_root_label'], grid_search_cv.best_par
```

```
C Parameter: 0.001, Cross-Validation Accuracy: 0.5028776978417266
C Parameter: 0.01, Cross-Validation Accuracy: 0.7848920863309352
C Parameter: 0.1, Cross-Validation Accuracy: 0.9384892086330936
C Parameter: 1, Cross-Validation Accuracy: 0.9507194244604318
C Parameter: 10, Cross-Validation Accuracy: 0.9546762589928057
C Parameter: 100, Cross-Validation Accuracy: 0.9550359712230214
C Parameter: 1000, Cross-Validation Accuracy: 0.9532374100719425
C Parameter: 10000, Cross-Validation Accuracy: 0.9532374100719425
C Parameter: 100000, Cross-Validation Accuracy: 0.9528776978417266
C Parameter: 1000000, Cross-Validation Accuracy: 0.9528776978417266

Optimal C Parameter:  100
```




```
C = 100
Accuracy: 0.9583333333333334
Recall: 0.9592391304347826
Precision: 0.9618528610354223
F-1 Score: 0.9605442176870749
```



5b-answer:

The best Gamma is 100

the result of this parameter is:

Accuracy: 0.9583333333333334

Recall: 0.9592391304347826

Precision: 0.9618528610354223

F-1 Score: 0.9605442176870749

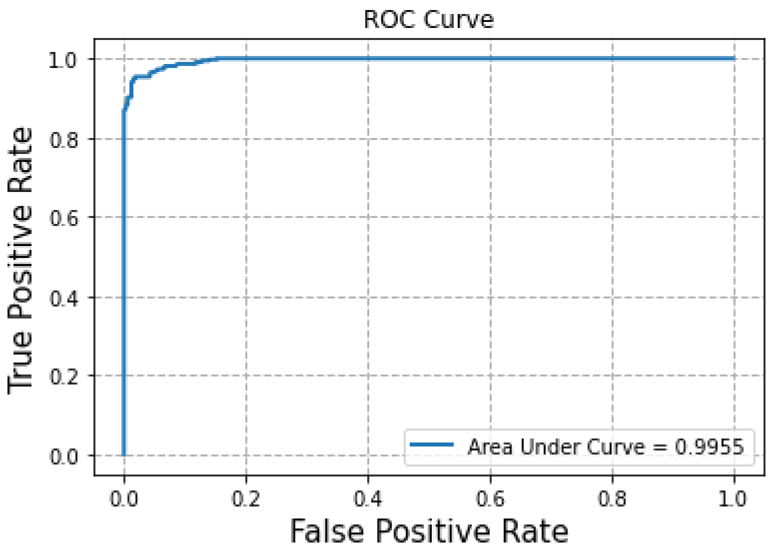
Question 6

- 6a.

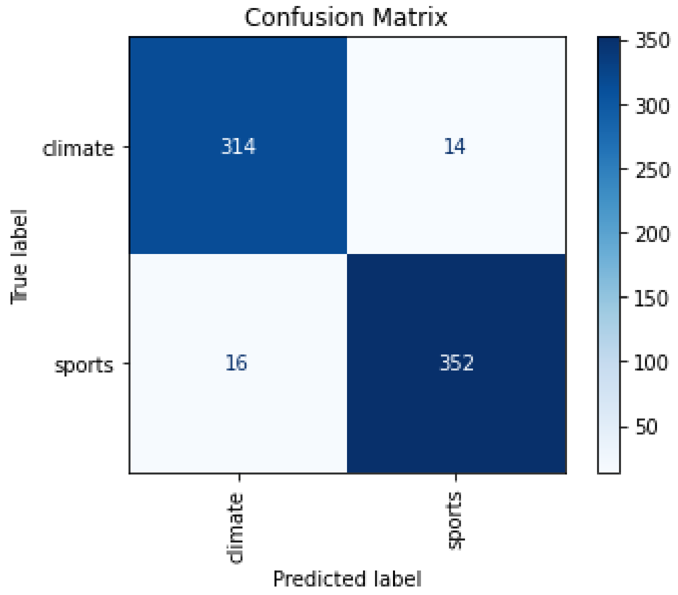
In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

def plot_roc_curve(probabilities, test_labels):
    fpr, tpr, _ = metrics.roc_curve(test_labels, probabilities)
    roc_auc = metrics.auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label='Area Under Curve = %0.4f' % roc_auc)
    plt.grid(color='0.7', linestyle='--', linewidth=1)
    plt.title('ROC Curve')
    plt.xlabel('False Positive Rate', fontsize=15)
    plt.ylabel('True Positive Rate', fontsize=15)
    plt.legend(loc="lower right")
    plt.show()

def print_classification_metrics(predictions, test_labels, label_names):
    print('Accuracy: ', metrics.accuracy_score(test_labels, predictions))
    print('Recall: ', metrics.recall_score(test_labels, predictions))
    print('Precision: ', metrics.precision_score(test_labels, predictions))
    print('F-1 Score: ', metrics.f1_score(test_labels, predictions))

def plot_confusion_matrix(confusion_matrix, label_names):
    display = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=label_names)
    display.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
    plt.title('Confusion Matrix')

def train_and_evaluate_logistic_regression(regularization, regularization_strength, train_data, train_labels, test_data, test_labe
    logistic_model = LogisticRegression(penalty=regularization, C=regularization_strength, random_state=42, solver='saga', max_ite
    logistic_model.fit(train_data, train_labels)

    # Plot ROC curve
    probabilities = logistic_model.decision_function(test_data)
    plot_roc_curve(probabilities, test_labels)

    # Print classification metrics
```

```
    predictions = logistic_model.predict(test_data)
    print_classification_metrics(predictions, test_labels, label_names)

    # Plot confusion matrix
    confusion_matrix = metrics.confusion_matrix(test_labels, predictions)
    plot_confusion_matrix(confusion_matrix, label_names)

# Train Logistic Regression with no regularization
train_and_evaluate_logistic_regression('none', 1.0, train_truncated, train['binary_root_label'], test_truncated, test['binary_root
```




```
Accuracy:  0.9568965517241379
Recall:  0.9565217391304348
Precision:  0.9617486338797814
F-1 Score:  0.9591280653950953
```



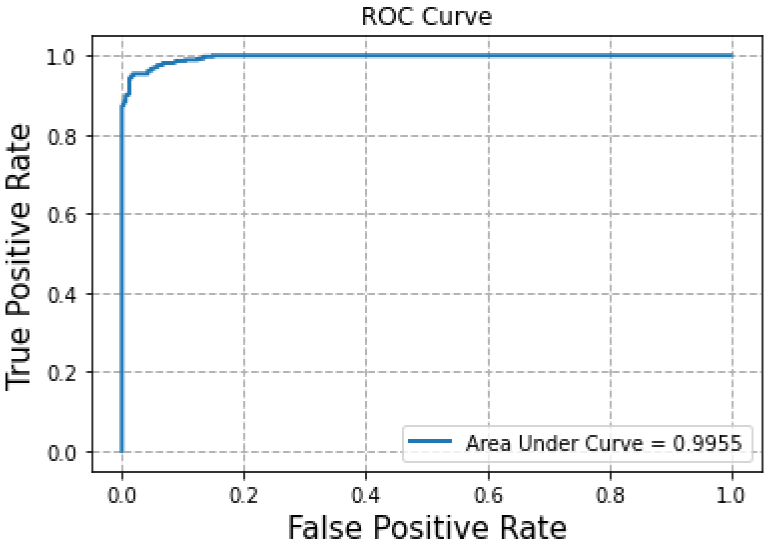
6b.1

In [ ]:

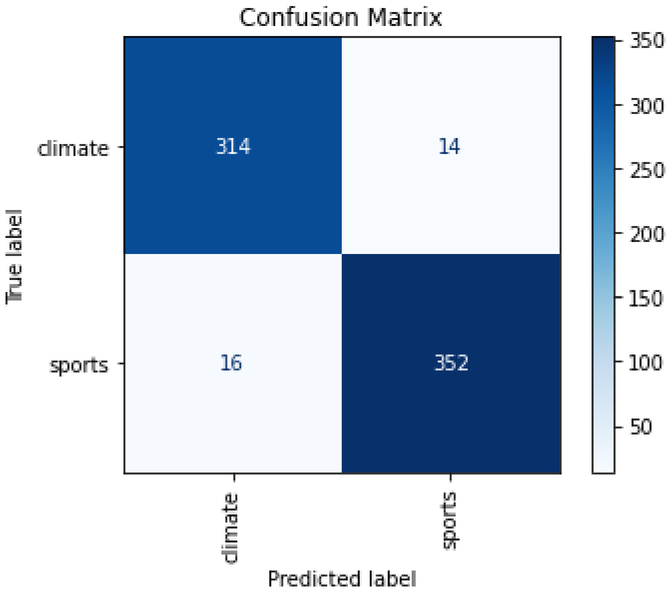
```
# Find best C value for L1 Regularization
param_grid = {'C': [10**(-5), 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]}
grid_l1 = GridSearchCV(LogisticRegression(penalty='l1', random_state=42, solver='saga', max_iter=100000), param_grid, cv=5, refit=
grid_l1.fit(train_truncated, train['binary_root_label'])

print("L1 Regularization Results")
c_values = [10**(-5), 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000]
for i in range(len(c_values)):
    print(f'C: {c_values[i]}, Cross Validation accuracy scores: {grid_l1.cv_results_["mean_test_score"][i]}')

print("\nBest C Value: ", grid_l1.best_params_['C'])
train_and_evaluate_logistic_regression('l1', grid_l1.best_params_['C'], train_truncated, train['binary_root_label'], test_truncate
```

```
L1 Regularization Results
C: 1e-05, Cross Validation accuracy scores: 0.5014388489208633
C: 0.0001, Cross Validation accuracy scores: 0.5014388489208633
C: 0.001, Cross Validation accuracy scores: 0.5014388489208633
C: 0.01, Cross Validation accuracy scores: 0.5014388489208633
C: 0.1, Cross Validation accuracy scores: 0.9262589928057554
C: 1, Cross Validation accuracy scores: 0.9496402877697842
C: 10, Cross Validation accuracy scores: 0.9535971223021582
C: 100, Cross Validation accuracy scores: 0.9543165467625899
```

```
C: 1000, Cross Validation accuracy scores: 0.9532374100719423
C: 10000, Cross Validation accuracy scores: 0.9532374100719423
C: 100000, Cross Validation accuracy scores: 0.9532374100719423

Best C Value:  100
```




```
Accuracy:  0.9568965517241379
Recall:  0.9565217391304348
Precision:  0.9617486338797814
F-1 Score:  0.9591280653950953
```

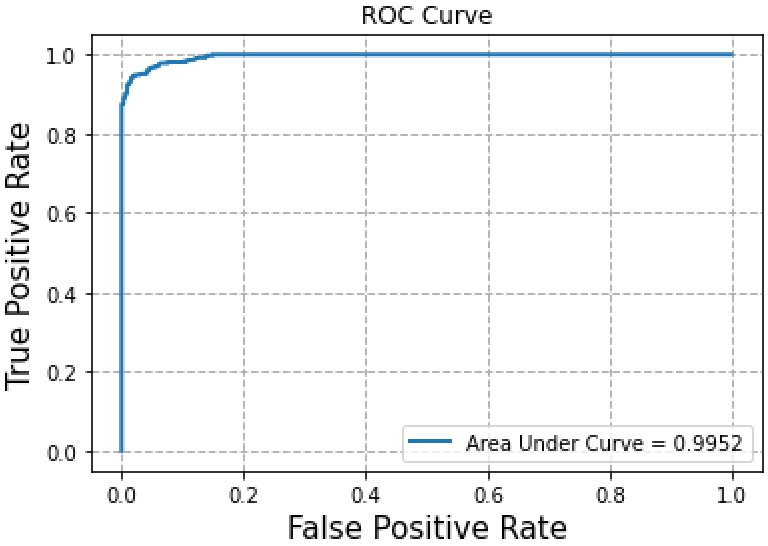


In [ ]:

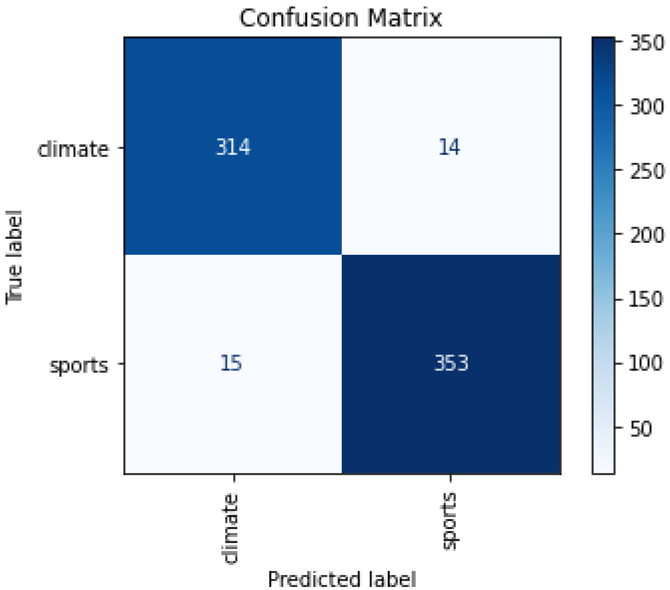
```
# Find best C value for L2 Regularization
grid_l2 = GridSearchCV(LogisticRegression(penalty='l2', random_state=42, solver='saga', max_iter=100000), param_grid, cv=5, refit=
grid_l2.fit(train_truncated, train['binary_root_label'])

print("L2 Regularization Results")
for i in range(len(c_values)):
    print(f'C: {c_values[i]}, Cross Validation accuracy scores: {grid_l2.cv_results_["mean_test_score"][i]}')

print("\nBest C Value: ", grid_l2.best_params_['C'])
train_and_evaluate_logistic_regression('l2', grid_l2.best_params_['C'], train_truncated, train['binary_root_label'], test_truncate
```

```
L2 Regularization Results
C: 1e-05, Cross Validation accuracy scores: 0.5028776978417266
C: 0.0001, Cross Validation accuracy scores: 0.5028776978417266
C: 0.001, Cross Validation accuracy scores: 0.8201438848920862
C: 0.01, Cross Validation accuracy scores: 0.94136690647482
C: 0.1, Cross Validation accuracy scores: 0.94136690647482
C: 1, Cross Validation accuracy scores: 0.9496402877697842
C: 10, Cross Validation accuracy scores: 0.9546762589928057
C: 100, Cross Validation accuracy scores: 0.9553956834532373
C: 1000, Cross Validation accuracy scores: 0.9543165467625899
C: 10000, Cross Validation accuracy scores: 0.9532374100719423
C: 100000, Cross Validation accuracy scores: 0.9532374100719423
```

```
Best C Value:  100
```




```
Accuracy:  0.9583333333333334
Recall:  0.9592391304347826
Precision:  0.9618528610354223
F-1 Score:  0.9605442176870749
```



6b.2

- Shown below are the best accuracy, recall, precision, and F-1 scores achieved by regression models with different regularizations. The best regularization strength for L1 and L2 regularization were found to be 100 and 100, respectively. Examination of the table reveals that the logistic regressions, optimized with their best C parameters, exhibited nearly identical performance metrics for No Regularization and L1. Moreover, both underperformd the L2 regularization by a slight margin.

Regularization Type	Accuracy	Recall	Precision	F-1 Score
No Regularization	0.9568965517241379	0.9565217391304348	0.9617486338797814	0.9591280653950953
L1	0.9568965517241379	0.9565217391304348	0.9617486338797814	0.9591280653950953
L2	0.9583333333333334	0.9592391304347826	0.9618528610354223	0.9605442176870749

6b.3

- How does the regularization parameter affect the test error?

Regularization is mainly used to prevent model overfitting. The use of regularization parameters is mainly to make the model less complex, thereby reducing the risk of overfitting and improving performance. At the same time, when the regularization parameters are too large, The model becomes too simple and cannot capture the key information in the data well, which may increase the test error.

- How are the learnt coefficients affected?

L1 regularization mainly produces a sparse model, in which many coefficients become 0, and the model will only use a part of the features, which means that such regularization helps in feature selection. L2 regularization can mainly solve the problem when the correlation of features is too high or the noise is large, it helps to control the complexity of the model, thereby improving the performance of the model.

- Why might one be interested in each type of regularization?

When the data has many irrelevant features, people can use L1 regularization to help simplify the model and feature selection. When the correlation of the data features is high or the nosie is too large,L2 regularization will help control the complexity of the model, thereby improving the performance of the model.People can apply different regularization methods according to different needs to improve the performance of the model.

6b.4

- The linear SVM utilizes a hyperplane as the decision boundary, aiming to maximize the distance between different classes in the data. Specifically, it seeks to find a linear decision boundary that optimally separates these classes. On the other hand, logistic regression adjusts the decision boundary to maximize the likelihood estimation. Unlike the SVM, logistic regression focuses on the probability of a data point belonging to a specific class rather than the distance between classes and the boundary. Their performance diverges due to their distinct approaches in determining the decision boundary. \ Moreover, SVM, by emphasizing the separation between classes, exhibits lower susceptibility to overfitting compared to logistic regression. The performance contrast is further accentuated by the probabilistic nature of logistic regression and the more deterministic nature of SVM. While they often demonstrate similar performance in comparable tasks, the disparity becomes statistically significant when applied to markedly different tasks. SVM may be preferred for tasks like image classification or text recognition, whereas logistic regression might be more suitable for scenarios where class separation is challenging.

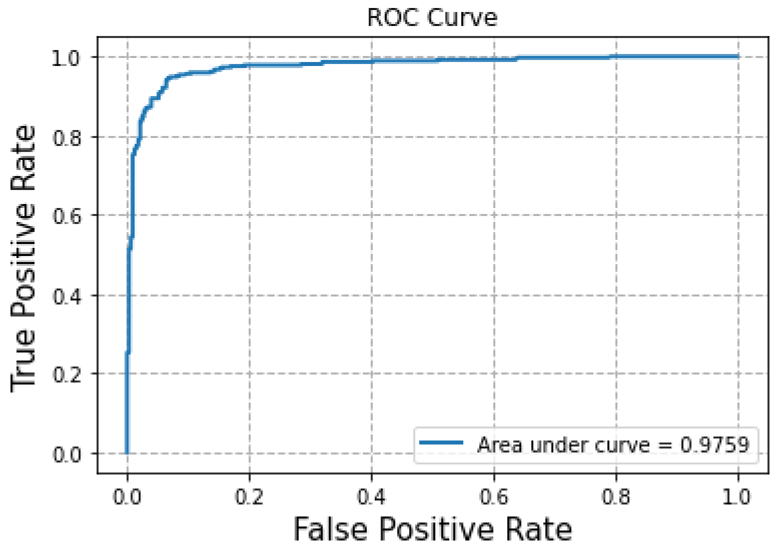
Question 7

In [ ]:

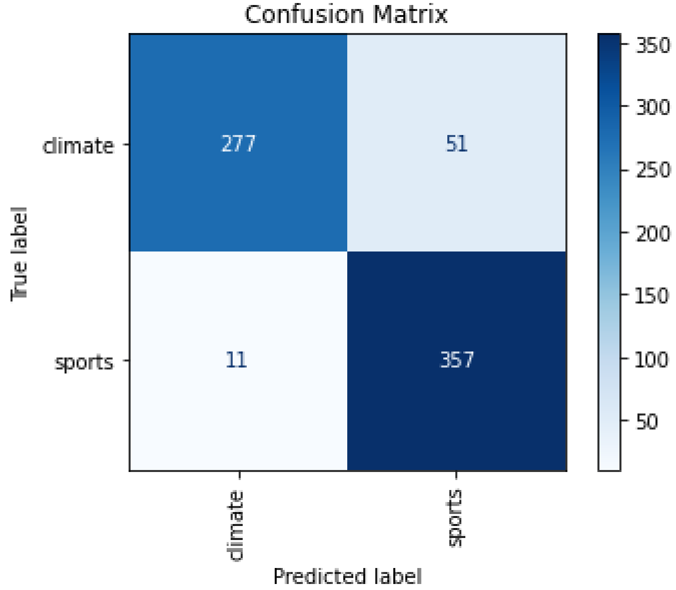
```
from sklearn.naive_bayes import GaussianNB

# Train Gaussian Naive Bayes classifier, print metrics, and plot ROC and confusion matrix
def train_gaussian_nb(train_data, train_labels, test_data, test_labels, label_names):
    gnb_classifier = GaussianNB()
    gnb_classifier.fit(train_data, train_labels)

    # Plot ROC curve
    probabilities = gnb_classifier.predict_proba(test_data)
    fpr, tpr, _ = metrics.roc_curve(test_labels, probabilities[:, 1])
    roc_auc = metrics.auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label='Area under curve = %0.4f' % roc_auc)
    plt.grid(color='0.7', linestyle='--', linewidth=1)
    plt.title('ROC Curve')
    plt.xlabel('False Positive Rate', fontsize=15)
    plt.ylabel('True Positive Rate', fontsize=15)
    plt.legend(loc="lower right")
    plt.show()

    # Print classification metrics
    predictions = gnb_classifier.predict(test_data)
    print('Accuracy: ', metrics.accuracy_score(test_labels, predictions))
    print('Recall: ', metrics.recall_score(test_labels, predictions))
    print('Precision: ', metrics.precision_score(test_labels, predictions))
    print('F-1 Score: ', metrics.f1_score(test_labels, predictions))

    # Plot confusion matrix
    confusion_matrix = metrics.confusion_matrix(test_labels, predictions)
    display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=label_names)
    display.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
    plt.title('Confusion Matrix')
```

In [ ]:

```
train_gaussian_nb(train_truncated, train['binary_root_label'], test_truncated, test['binary_root_label'], categories)
```




```
Accuracy:  0.9109195402298851
Recall:  0.970108695652174
Precision:  0.875
F-1 Score:  0.9201030927835052
```



Grid Search of Parameters

Question 8

In [ ]:

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

def stem_tokenize(text):
    return [stemmer.stem(token) for token in nltk.word_tokenize(text)]
```

In [ ]:

```
import time
from sklearn.pipeline import Pipeline

full_time = time.time()
from shutil import rmtree

pipe = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('dim_red', TruncatedSVD()),
    ('classifier', SVC())
])

param_grid = [
    {
        # feature extraction
        'vect__min_df': [3, 5],
        'vect__tokenizer': [None],  # default lemmatization
        'vect__analyzer': [stem_rmv_punc],  # if lemmatization
        'vect__stop_words':['english'],
        # dimensionality Reduction
        'dim_red': [TruncatedSVD()],
        'dim_red__n_components': [5, 30, 80],
        # Classifier
        'classifier': [LogisticRegression(penalty='l1', max_iter=100000, solver='saga',C = 100), LogisticRegression(penalty='l2',
                    SVC(C=100), GaussianNB()]
    },
    {
        # feature extraction
        'vect__min_df': [3, 5],
        'vect__tokenizer': [stem_tokenize],  # if stemming
```

```python
        'vect__analyzer': ['word'],  # default analyzer
        'vect__stop_words':['english'],
        # dimensionality Reduction
        'dim_red': [TruncatedSVD()],
        'dim_red__n_components': [5, 30, 80],
        # Classifier
        'classifier': [LogisticRegression(penalty='l1', max_iter=100000, solver='saga',C = 100), LogisticRegression(penalty='l2',
                    SVC(C=100), GaussianNB()]
    },
    {
        # feature extraction
        'vect__min_df': [3, 5],
        'vect__tokenizer': [None],  # both None and stem_tokenize
        'vect__analyzer': [stem_rmv_punc],  # both stem_rmv_punc and default analyzer
        'vect__stop_words': ['english'],
        # dimensionality reduction
        'dim_red': [NMF(init='random', random_state=0)],
        'dim_red__n_components': [5, 30, 80],
        'dim_red__init': ['random'],  # NMF-specific parameters
        'dim_red__random_state': [0],
        # Classifier
        'classifier': [LogisticRegression(penalty='l1', max_iter=100000, solver='saga', C=100),
                       LogisticRegression(penalty='l2', max_iter=100000, solver='saga', C=100),
                       SVC(C=100), GaussianNB()]
    },
    # Grid for NMF
    {
        # feature extraction
        'vect__min_df': [3, 5],
        'vect__tokenizer': [stem_tokenize],
        'vect__analyzer': ['word'],
        'vect__stop_words': ['english'],
        # dimensionality reduction
        'dim_red': [NMF(init='random', random_state=0)],
        'dim_red__n_components': [5, 30, 80],
        'dim_red__init': ['random'],  # NMF-specific parameters
        'dim_red__random_state': [0],
        # Classifier
        'classifier': [LogisticRegression(penalty='l1', max_iter=100000, solver='saga', C=100),
                       LogisticRegression(penalty='l2', max_iter=100000, solver='saga', C=100),
                       SVC(C=100), GaussianNB()]
    }

]
```

```
# GridSearchCV
search = GridSearchCV(pipe,param_grid, n_jobs=8, cv=5,scoring='accuracy')
search.fit(train['cleaned_text'], train['binary_root_label'])

print(f"--- Total time {time.time() - full_time} seconds ---")
```

```
--- Total time 1581.886403799057 seconds ---
```

In [ ]:

```
results_df = pd.DataFrame(search.cv_results_)
top_5 = results_df.sort_values(by='mean_test_score', ascending=False).head(5)
```

In [ ]:

```
with open('output.txt', 'w') as f:
    f.write(results_df.to_string())
```

In [ ]:

```
top_5
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_classifier	param_dim_red	param_dim_red__n_components	
16	23.596305	0.254585	5.920521	0.129643	SVC(C=100)	TruncatedSVD(n_components=80)	80	C
17	23.697774	0.128029	5.863749	0.156670	SVC(C=100)	TruncatedSVD(n_components=80)	80	C
41	13.679450	0.230603	3.456381	0.065684	SVC(C=100)	TruncatedSVD()	80	
11	23.833757	0.152688	5.925853	0.139345	LogisticRegression(C=100, max_iter=100000, sol...	TruncatedSVD(n_components=80)	80	C
64	32.886108	1.981341	6.536150	0.268975	SVC(C=100)	NMF(init='random', random_state=0)	80	C

5 rows × 22 columns

In [ ]:

```
top_5['params'].index
```

Out[ ]:

```
Int64Index([16, 17, 41, 11, 64], dtype='int64')
```

In [ ]:

```
for i in top_5['params'].index:
    print('-' * 20)
    print(top_5['params'][i])
```

```
--------------------
{'classifier': SVC(C=100), 'dim_red': TruncatedSVD(n_components=80), 'dim_red__n_components': 80, 'vect__analyzer': <function stem_
rmv_punc at 0x000002A8C8FCFE18>, 'vect__min_df': 3, 'vect__stop_words': 'english', 'vect__tokenizer': None}
--------------------
{'classifier': SVC(C=100), 'dim_red': TruncatedSVD(n_components=80), 'dim_red__n_components': 80, 'vect__analyzer': <function stem_
rmv_punc at 0x000002A8C8FCFE18>, 'vect__min_df': 5, 'vect__stop_words': 'english', 'vect__tokenizer': None}
--------------------
{'classifier': SVC(C=100), 'dim_red': TruncatedSVD(), 'dim_red__n_components': 80, 'vect__analyzer': 'word', 'vect__min_df': 5, 've
ct__stop_words': 'english', 'vect__tokenizer': <function stem_tokenize at 0x000002A991501D08>}
--------------------
{'classifier': LogisticRegression(C=100, max_iter=100000, solver='saga'), 'dim_red': TruncatedSVD(n_components=80), 'dim_red__n_com
ponents': 80, 'vect__analyzer': <function stem_rmv_punc at 0x000002A8C8FCFE18>, 'vect__min_df': 5, 'vect__stop_words': 'english',
'vect__tokenizer': None}
--------------------
{'classifier': SVC(C=100), 'dim_red': NMF(init='random', random_state=0), 'dim_red__init': 'random', 'dim_red__n_components': 80,
'dim_red__random_state': 0, 'vect__analyzer': <function stem_rmv_punc at 0x000002A8C8FCFE18>, 'vect__min_df': 3, 'vect__stop_word
s': 'english', 'vect__tokenizer': None}
```

In [ ]:

```
for index, row in top_5.iterrows():
    # Setup the pipeline with the best parameters
    pipe.set_params(**row['params'])

    # Fit the pipeline on the entire training set
    pipe.fit(train['cleaned_text'], train['binary_root_label'])

    # Evaluate on the test set
    prediction_bestpip = pipe.predict(test['cleaned_text'])

    print('-' * 20)
    print('Accuracy: ', metrics.accuracy_score(test['binary_root_label'], prediction_bestpip))
    print('Recall: ', metrics.recall_score(test['binary_root_label'], prediction_bestpip))
    print('Precision: ', metrics.precision_score(test['binary_root_label'], prediction_bestpip))
    print('F-1 Score: ', metrics.f1_score(test['binary_root_label'], prediction_bestpip))
```

```
--------------------
Accuracy:  0.9770114942528736
Recall:  0.9755434782608695
Precision:  0.9808743169398907
F-1 Score:  0.9782016348773842
--------------------
Accuracy:  0.9683908045977011
Recall:  0.9592391304347826
Precision:  0.9805555555555555
F-1 Score:  0.9697802197802197
--------------------
Accuracy:  0.9626436781609196
Recall:  0.9592391304347826
Precision:  0.9697802197802198
F-1 Score:  0.9644808743169399
--------------------
Accuracy:  0.9669540229885057
Recall:  0.9619565217391305
Precision:  0.9752066115702479
F-1 Score:  0.9685362517099864
--------------------
Accuracy:  0.9741379310344828
Recall:  0.9646739130434783
Precision:  0.9861111111111112
F-1 Score:  0.9752747252747254
```

Q8-answer

Top 5 Combinations

NO.1: 'classifier': SVC(C=100), 'dim_red': TruncatedSVD(n_components=80)-LSI, 'dim_red**n_components': 80, Lemmatization, 'vect**min_df': 3'

- **Accuracy:** 0.9770114942528736
- **Recall:** 0.9755434782608695
- **Precision:** 0.9808743169398907
- **F-1 Score:** 0.9782016348773842

No.2: 'classifier': SVC(C=100), 'dim_red': TruncatedSVD(n_components=80)-LSI, 'dim_red**n_components': 80, Lemmatization ,'vect**min_df': 5

- **Accuracy:** 0.9683908045977011
- **Recall:** 0.9592391304347826
- **Precision:** 0.9805555555555555
- **F-1 Score:** 0.9697802197802197

No.3: {'classifier': SVC(C=100), 'dim_red': TruncatedSVD()-LSI, 'dim_red**n_components': 80, Stemming, 'vect**min_df': 5}

- **Accuracy:** 0.9626436781609196
- **Recall:** 0.9592391304347826
- **Precision:** 0.9697802197802198
- **F-1 Score:** 0.9644808743169399

No.4: {'classifier': LogisticRegression(penalty='l2',C=100, max_iter=100000, solver='saga'), 'dim_red': TruncatedSVD(n_components=80)-LSI, 'dim_red**n_components': 80, Lemmatization, 'vect**min_df': 5}

- **Accuracy:** 0.9669540229885057
- **Recall:** 0.9619565217391305
- **Precision:** 0.9752066115702479
- **F-1 Score:** 0.9685362517099864

No.5:{'classifier': SVC(C=100), 'dim_red': NMF(init='random', random_state=0),'dim_red**n_components':80, Lemmatization, 'vect**min_df': 3}

- **Accuracy:** 0.9741379310344828
- **Recall:** 0.9646739130434783
- **Precision:** 0.9861111111111112
- **F-1 Score:** 0.9752747252747254

All result of Gridsearchcv shown in output.txt

Multiclass Classification

Questation 9

In [ ]:

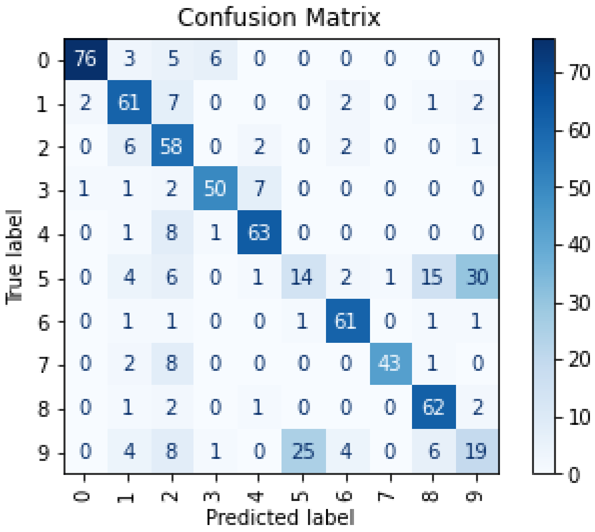
```
np.random.seed(42)
random.seed(42)

train_mult, test_mult = train_test_split(df[["full_text", "leaf_label"]], test_size=0.2, random_state=42)

# data preprocess
train_mult['cleaned_text'] = train_mult['full_text'].apply(clean)
test_mult['cleaned_text'] = test_mult['full_text'].apply(clean)
```

```python
X_train_tfidf_mult = tfidf.fit_transform(count_vectorizer.fit_transform(train_mult['cleaned_text']))
X_test_tfidf_mult = tfidf.transform(count_vectorizer.transform(test_mult['cleaned_text']))

map_row_to_class = {
    0: "basketball", 1: "baseball", 2: "tennis",
    3: "football", 4: "soccer", 5: "forest fire",
    6: "flood", 7: "earthquake", 8: "drought", 9: "heatwave"
}

class_to_num = {v: k for k, v in map_row_to_class.items()}
test_mult['binary_root_label'] = [class_to_num[label] for label in test_mult['leaf_label']]
train_mult['binary_root_label'] = [class_to_num[label] for label in train_mult['leaf_label']]

svd_components = 80
truncated_svd_mult = TruncatedSVD(n_components=svd_components, random_state=42)
train_truncated_mult = truncated_svd.fit_transform(X_train_tfidf_mult)
test_truncated_mult = truncated_svd.transform(X_test_tfidf_mult)


# mult_classifer
from sklearn.naive_bayes import GaussianNB
gnb_classifier = GaussianNB()
gnb_classifier.fit(train_truncated_mult, train_mult['binary_root_label'])

predictions = gnb_classifier.predict(test_truncated_mult)
```

In [ ]:

```python
confusion_matrix = metrics.confusion_matrix(test_mult['binary_root_label'] , predictions)
confusion_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=np.unique(test_mult['binary_r
confusion_display.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
plt.title('Confusion Matrix')
```

Out[ ]: Text(0.5, 1.0, 'Confusion Matrix')



In [ ]:

```
print('Accuracy: ', metrics.accuracy_score(test_mult['binary_root_label'], predictions))
print('Recall: ', metrics.recall_score(test_mult['binary_root_label'], predictions, average='macro'))
print('Precision: ', metrics.precision_score(test_mult['binary_root_label'], predictions, average='macro'))
print('F-1 Score: ', metrics.f1_score(test_mult['binary_root_label'], predictions, average='macro'))
```

```
Accuracy:  0.728448275862069
Recall:  0.7288709655591503
Precision:  0.7206829496758594
F-1 Score:  0.7163369334734864
```

In [ ]:

```
from sklearn.multiclass import OneVsOneClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.utils.class_weight import compute_class_weight

def train_evaluate_SVM_ovo(train_data, train_labels, test_data, test_labels, C_value):
    # Train SVM
    # clf_ovo = OneVsOneClassifier(SVC(C=C_value, class_weight='balanced'))
    clf_ovo = OneVsOneClassifier(SVC(C=C_value))
    clf_ovo.fit(train_data, train_labels)

    # Print SVM metrics
    print(f"C = {C_value}")
    predictions = clf_ovo.predict(test_data)
    print('Accuracy: ', metrics.accuracy_score(test_labels, predictions))
    print('Recall: ', metrics.recall_score(test_labels, predictions, average='weighted'))
```

```python
    print('Precision: ', metrics.precision_score(test_labels, predictions, average='weighted'))
    print('F-1 Score: ', metrics.f1_score(test_labels, predictions, average='weighted'))

    # Plot confusion matrix
    confusion_matrix = metrics.confusion_matrix(test_labels, predictions)
    confusion_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=np.unique(test_labels))
    confusion_display.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
    plt.title('Confusion Matrix')

def train_evaluate_SVM_ovr(train_data, train_labels, test_data, test_labels, C_value):
    # Train SVM
    # clf_ovr = OneVsRestClassifier(SVC(C=C_value, class_weight='balanced'))
    clf_ovr = OneVsRestClassifier(SVC(C=C_value))
    clf_ovr.fit(train_data, train_labels)

    # Print SVM metrics
    print(f"C = {C_value}")
    predictions = clf_ovr.predict(test_data)
    print('Accuracy: ', metrics.accuracy_score(test_labels, predictions))
    print('Recall: ', metrics.recall_score(test_labels, predictions, average='weighted'))
    print('Precision: ', metrics.precision_score(test_labels, predictions, average='weighted'))
    print('F-1 Score: ', metrics.f1_score(test_labels, predictions, average='weighted'))

    # Plot confusion matrix
    confusion_matrix = metrics.confusion_matrix(test_labels, predictions)
    confusion_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=np.unique(test_labels))
    confusion_display.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
    plt.title('Confusion Matrix')
```

In [ ]:

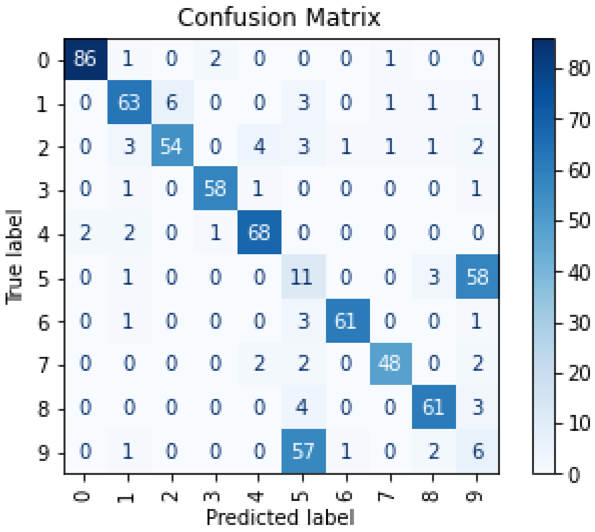
```python
def train_evaluate_SVM_ovo_solveci(train_data, train_labels, test_data, test_labels, C_value):
    # Train SVM
    clf_ovo = OneVsOneClassifier(SVC(C=C_value, class_weight='balanced'))
    clf_ovo.fit(train_data, train_labels)

    # Print SVM metrics
    print(f"C = {C_value}")
    predictions = clf_ovo.predict(test_data)
    print('Accuracy: ', metrics.accuracy_score(test_labels, predictions))
    print('Recall: ', metrics.recall_score(test_labels, predictions, average='weighted'))
    print('Precision: ', metrics.precision_score(test_labels, predictions, average='weighted'))
    print('F-1 Score: ', metrics.f1_score(test_labels, predictions, average='weighted'))

    # Plot confusion matrix
```

```python
    confusion_matrix = metrics.confusion_matrix(test_labels, predictions)
    confusion_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=np.unique(test_labels))
    confusion_display.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
    plt.title('Confusion Matrix')

def train_evaluate_SVM_ovr_solveci(train_data, train_labels, test_data, test_labels, C_value):
    # Train SVM
    clf_ovr = OneVsRestClassifier(SVC(C=C_value, class_weight='balanced'))
    clf_ovr.fit(train_data, train_labels)

    # Print SVM metrics
    print(f"C = {C_value}")
    predictions = clf_ovr.predict(test_data)
    print('Accuracy: ', metrics.accuracy_score(test_labels, predictions))
    print('Recall: ', metrics.recall_score(test_labels, predictions, average='weighted'))
    print('Precision: ', metrics.precision_score(test_labels, predictions, average='weighted'))
    print('F-1 Score: ', metrics.f1_score(test_labels, predictions, average='weighted'))

    # Plot confusion matrix
    confusion_matrix = metrics.confusion_matrix(test_labels, predictions)
    confusion_display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix, display_labels=np.unique(test_labels))
    confusion_display.plot(cmap=plt.cm.Blues, xticks_rotation='vertical')
    plt.title('Confusion Matrix')
```

In [ ]:

```python
# result of one vs one
train_evaluate_SVM_ovo_solveci(train_truncated_mult, train_mult['binary_root_label'], test_truncated_mult, test_mult['binary_root_
```

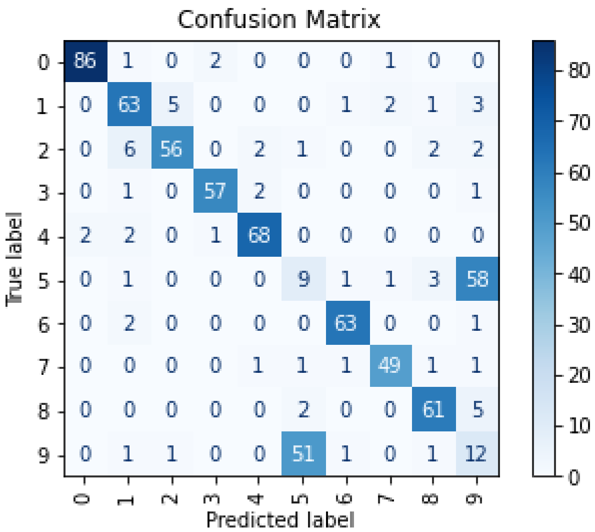
```
C = 100
Accuracy:  0.7413793103448276
Recall:  0.7413793103448276
Precision:  0.7612107213543277
F-1 Score:  0.7506511481390324
```



In [ ]:

```
# result of one vs rest
train_evaluate_SVM_ovr_solveci(train_truncated_mult, train_mult['binary_root_label'], test_truncated_mult, test_mult['binary_root_
```

```
C = 100
Accuracy:  0.7528735632183908
Recall:  0.7528735632183908
Precision:  0.7609817618889433
F-1 Score:  0.7563951590157423
```



The result that after merge 9 to 5

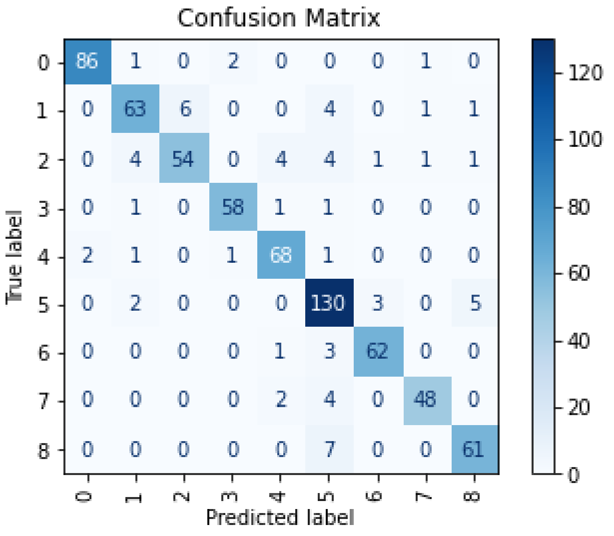
In [ ]:

```
# Solve class imbalance issue
train_mult['changed_lable'] = 0
test_mult['changed_lable'] = 0
train_mult['changed_lable'] = train_mult['binary_root_label'].replace(9, 5)
test_mult['changed_lable'] = test_mult['binary_root_label'].replace(9, 5)
```

In [ ]:

```
# result of one vs one(solved class imbalance issue)
train_evaluate_SVM_ovo(train_truncated_mult, train_mult['changed_lable'], test_truncated_mult, test_mult['changed_lable'], 100)
```

```
C = 100
Accuracy:  0.9051724137931034
Recall:  0.9051724137931034
Precision:  0.9066101760370868
F-1 Score:  0.9049299901511274
```

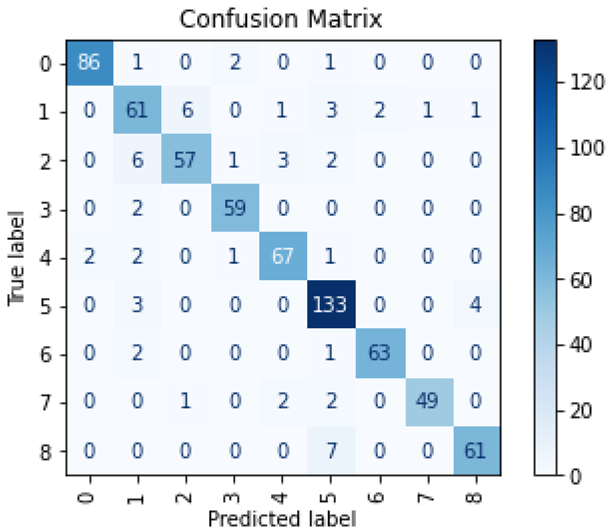


In [ ]:

```
# result of one vs rest(solved class imbalance issue)
train_evaluate_SVM_ovr(train_truncated_mult, train_mult['changed_lable'], test_truncated_mult, test_mult['changed_lable'], 100)
```

```
C = 100
Accuracy:  0.9137931034482759
Recall:  0.9137931034482759
```

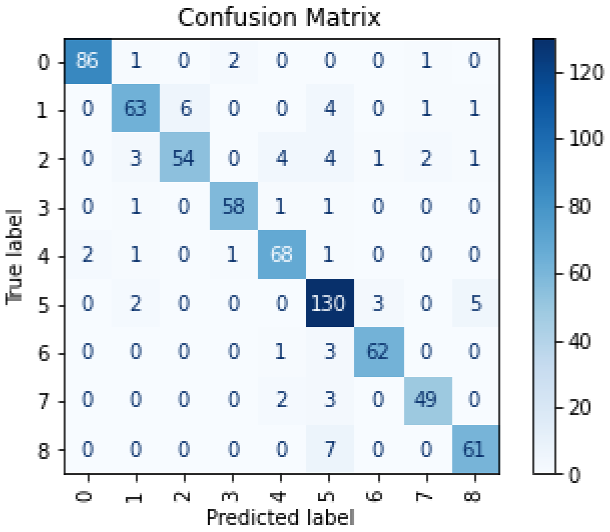
```
Precision:  0.9149731148680501
F-1 Score:  0.9138572946484304
```



The result that after merge 9 to 5-solve imbalance

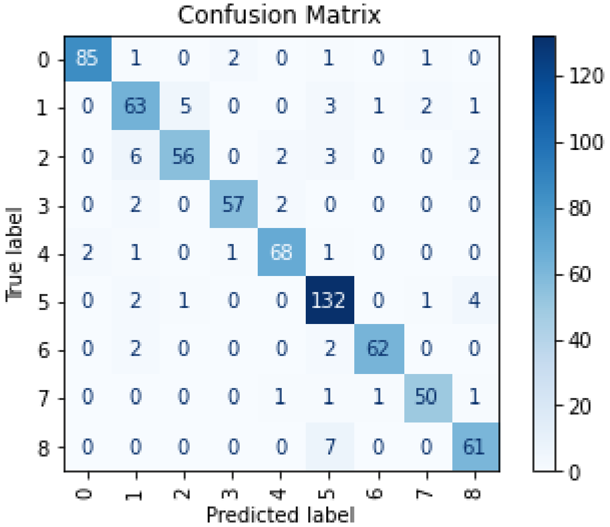
In [ ]:
```
train_evaluate_SVM_ovo_solveci(train_truncated_mult, train_mult['changed_lable'], test_truncated_mult, test_mult['changed_lable'],
```

```
C = 100
Accuracy:  0.9066091954022989
Recall:  0.9066091954022989
Precision:  0.907756332429278
F-1 Score:  0.9062940513107396
```



```
In [ ]: train_evaluate_SVM_ovr_solveci(train_truncated_mult, train_mult['changed_lable'], test_truncated_mult, test_mult['changed_lable'],
```

```
C = 100
Accuracy:  0.9109195402298851
Recall:  0.9109195402298851
Precision:  0.9120986975869851
F-1 Score:  0.9109655717406929
```



Q9-answer:

1 How to resolve class imbalance?

The imbalance issue can be addressed using classs_weight='balanced' parameter in initiation the One vs rest SVM classier. The assigned weight inversely proportional to class frequency, and address the issue.

2 Observation on confusion matrix:

Yes, there are visible blocks along major diagonal of the matrix. The blocks is indicating instances that classier made right predictions for each class. A strong diagonal means that classier is doing well. But for label 9 and label 5, the squares here are not very obvious. On the contrary, it can be clearly seen in the confusion matrix. Whether it is Na¨ive Bayes classification or SVM classification, the model will incorrectly classify label 5 into label 9, or label 9 is incorrectly classified as label 5, which means that the two labels are very similar.

3 suggest a subset of label:

I merge label 9(heatwave) to label 5(forest fire),and re computer the result , the result is better than the previous one, Accuracy, Recall, Precison adn F-1 Score is increased about 0.17.result is shown above.

4 class imbalance impact performance?

Yes, the class imbalance is impacting the performance after merging class. For example, when heatwave and forest fire into Natural disaster, the addressed class imbalance by using different classes weight during the training process, and then the result is that accuracy is changed.

Word Embedding

Q10

Question 10 text answer:

10a: GLOVE is tranined on ratio to capture the information about relations among words. Ratio is helpful to emphasis the inportance of word co-occurance, and this mades the result vector to have more information than the probablity itself.

10b: Certainly, they would produce identical embedding vectors for both scenarios. Although GLOVE is trained by considering the global context of each word, during inference, it generates embeddings by solely focusing on the individual tokens.

10c: The value of the difference between woman vs man, and wife vs hisband will be similar as they are differnt in gender, but the wife vs orange will be very different as they are totally different words.

- ||GLoVE["woman"] - GLoVE["man"]||2 ≈ ||GLoVE["wife"] - GLoVE["husband"]||2 < ||GLoVE["wife"] - GLoVE["orange"]||2

10d: GLOVE is trained using the full words, thus it is better to use lemmatization because this can retain the base form of words, and more suitable for GLOVE.

Q11

In [ ]:

```
embeddings_dict = {}
dimension_of_glove = 300
with open("glove.6B/glove.6B.300d.txt", 'rb') as f: # if 'r' fails with unicode error, please use 'rb'
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], "float32")
        embeddings_dict[word] = vector
```

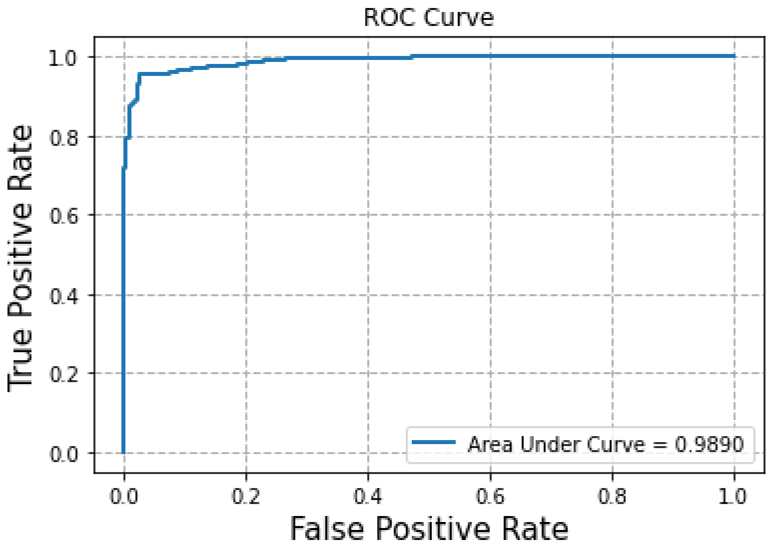
In [ ]:

```
from sklearn.preprocessing import LabelEncoder
import numpy as np
import random
import nltk
from nltk import word_tokenize
import ast

# Seed random generator
np.random.seed(42)
random.seed(42)

# Split data into testing and training and clean it
train_set, test_set = train_test_split(df[["root_label", "keywords", "full_text"]], test_size=0.2)

train_set['full_text'] = train_set['full_text'].apply(clean)
test_set['full_text'] = test_set['full_text'].apply(clean)

# Transform categories into number (Climate=0, Sports=1)
label_encoder = LabelEncoder()
train_set['binary_root_label'] = label_encoder.fit_transform(train_set['root_label'])
```

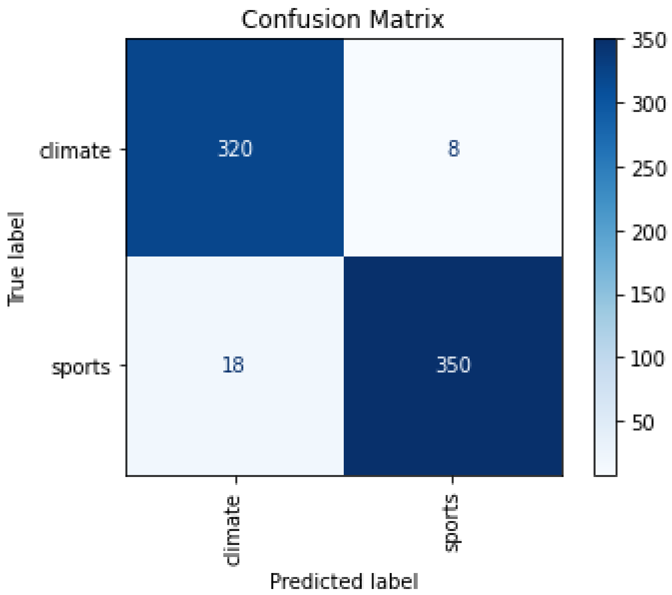
```python
test_set['binary_root_label'] = label_encoder.fit_transform(test_set['root_label'])

# Do the feature engineering process
def key_to_vec(keyword, embeddings_dict):
    average_embeddings = []

    for i in keyword.index:
        keyword_embeddings = []

        for word in re.sub(r'[^A-Za-z0-9]+', ' ', keyword[i]).split():
            byte_word = word.encode('utf-8')
            if byte_word in embeddings_dict:
                keyword_embeddings.append(embeddings_dict[byte_word])

        if keyword_embeddings:
            average_embedding = np.mean(keyword_embeddings, axis=0)
        else:
            average_embedding = np.zeros(300)

        average_embeddings.append(average_embedding)

    return average_embeddings


train_set['glove_data'] = 0
train_set['glove_data'] = 0

train_set['glove_data']  = key_to_vec(train_set['keywords'],embeddings_dict)
test_set['glove_data']  = key_to_vec(test_set['keywords'],embeddings_dict)

X = np.vstack(train_set['glove_data'].values)
y = train_set['binary_root_label'].values
X_test = np.vstack(test_set['glove_data'].values)
y_test = test_set['binary_root_label'].values
```

In [ ]:

```python
train_and_evaluate_logistic_regression('l1', 100, X, y, X_test, y_test, categories)
```




```
Accuracy:  0.9626436781609196
Recall:  0.9510869565217391
Precision:  0.9776536312849162
F-1 Score:  0.9641873278236915
```



Question 11 text answer

11a:

A feature engineering process will be needed, and it will need to be firstly tockenized to make each documents into indivudual tockens. Then, words embedding loopup will be needed to look up the corresponding GLOVE ords vector for each document After that, normalization will be needed to normalize each word vector to ensure consistent scale and mitigate the impact of different docuemnt length. After that aggregation is needed to aggregate the normalized words vectors to form a single vector representation for the whlole docuemnt. The aggregation can be done using averaging the vectors. To aviod the curse of dimensionalty, dimensionality reduction can also be needed. Finally the resulting vector will be the final representation of the document using GLOVE.

Q12

In [ ]:

```
import numpy as np

def load_glove_model(file_path):
    embeddings_dict = {}
    with open(file_path, 'rb') as f: # if 'r' fails with unicode error, please use 'rb'
        for line in f:
            values = line.split()
            word = values[0]
            vector = np.asarray(values[1:], "float32")
            embeddings_dict[word] = vector
    return embeddings_dict

file_prefix = "glove.6B/"
files = ['glove.6B.50d.txt', 'glove.6B.100d.txt', 'glove.6B.200d.txt', 'glove.6B.300d.txt']

embeddings_dicts = [load_glove_model(file_prefix + file_name) for file_name in files]
```

In [ ]:

```
dimension_list = [50, 100, 200, 300]
X_train = {}
X_test = {}
y = train_set['binary_root_label'].values
y_test = test_set['binary_root_label'].values


for i, dim in enumerate(dimension_list):
    glove_data_key = f'glove_data_{dim}'
    train_set[glove_data_key] = key_to_vec(train_set['keywords'], embeddings_dicts[i])
```

```
    test_set[glove_data_key] = key_to_vec(test_set['keywords'], embeddings_dicts[i])


    X_train[dim] = np.vstack(train_set[glove_data_key].values)
    X_test[dim] = np.vstack(test_set[glove_data_key].values)

accuracies = []

Log = LogisticRegression(penalty='l1', C=1000, solver='saga', max_iter=100000)

for dim in dimension_list:
    Log.fit(X_train[dim], y)

    predictions = Log.predict(X_test[dim])

    accuracy = metrics.accuracy_score(y_test, predictions)
    accuracies.append(accuracy)
    print(f"Accuracy for {dim} dimensions: {accuracy}")
```

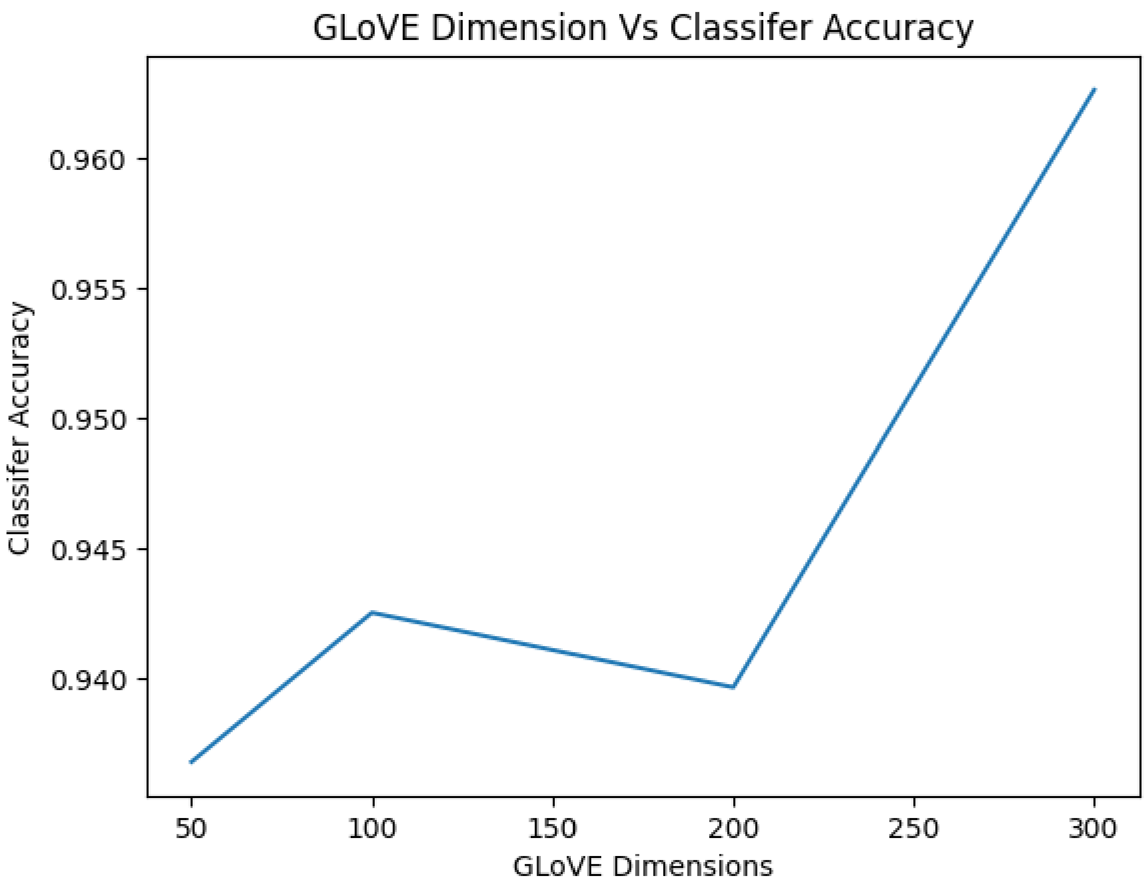
```
Accuracy for 50 dimensions: 0.9367816091954023
Accuracy for 100 dimensions: 0.9425287356321839
Accuracy for 200 dimensions: 0.9396551724137931
Accuracy for 300 dimensions: 0.9626436781609196
```

In [ ]:

```
plt.plot(dimension_list,accuracies)
plt.xlabel('GLoVE Dimensions')
plt.ylabel('Classifer Accuracy')
plt.title('GLoVE Dimension Vs Classifer Accuracy')
```

Out[ ]:

```
Text(0.5, 1.0, 'GLoVE Dimension Vs Classifer Accuracy')
```



Q12 text answer:

The trend is expected.

The plot shows when dimension increases, the accuracy overall increases, although this is not always the case, but the overall trend still shows this.This tells us that higher dimension representation allows better preservation of the semantic relational information and contact, producing better classification accuracy.

Q13

In [ ]:

```
!pip install umap-learn
!pip install umap-learn[plot]
```

```
Collecting umap-learn
  Downloading umap-learn-0.5.5.tar.gz (90 kB)
Requirement already satisfied: numpy>=1.17 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn) (1.19.5)
Requirement already satisfied: scipy>=1.3.1 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn) (1.5.4)
Requirement already satisfied: scikit-learn>=0.22 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn) (0.24.
2)
Collecting numba>=0.51.2
  Downloading numba-0.53.1-cp36-cp36m-win_amd64.whl (2.3 MB)
Collecting pynndescent>=0.5
  Downloading pynndescent-0.5.11-py3-none-any.whl (55 kB)
Requirement already satisfied: tqdm in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn) (4.64.1)
Requirement already satisfied: setuptools in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from numba>=0.51.2->umap-learn)
(58.0.4)
Collecting llvmlite<0.37,>=0.36.0rc1
  Downloading llvmlite-0.36.0-cp36-cp36m-win_amd64.whl (16.0 MB)
Requirement already satisfied: importlib-metadata>=4.8.1 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from pynndescent
>=0.5->umap-learn) (4.8.3)
Requirement already satisfied: joblib>=0.11 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from pynndescent>=0.5->umap-l
earn) (1.1.1)
Requirement already satisfied: typing-extensions>=3.6.4 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from importlib-me
tadata>=4.8.1->pynndescent>=0.5->umap-learn) (4.1.1)
Requirement already satisfied: zipp>=0.5 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from importlib-metadata>=4.8.1->
pynndescent>=0.5->umap-learn) (3.6.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from scikit-learn>=0.
22->umap-learn) (3.1.0)
Requirement already satisfied: colorama in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from tqdm->umap-learn) (0.4.5)
Requirement already satisfied: importlib-resources in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from tqdm->umap-learn)
(5.4.0)
Building wheels for collected packages: umap-learn
  Building wheel for umap-learn (setup.py): started
  Building wheel for umap-learn (setup.py): finished with status 'done'
  Created wheel for umap-learn: filename=umap_learn-0.5.5-py3-none-any.whl size=86853 sha256=489f770a3f0aff6dd69017de6bf56da23d4e4c
4bb6f9e0f0392f2279e3de610f
  Stored in directory: c:\users\michael_mi\appdata\local\pip\cache\wheels\fa\bf\ee\50cca124d5a1e2f411b47ff63c6dd5c0a152d9622e4c9944
9f
Successfully built umap-learn
Installing collected packages: llvmlite, numba, pynndescent, umap-learn
Successfully installed llvmlite-0.36.0 numba-0.53.1 pynndescent-0.5.11 umap-learn-0.5.5
Requirement already satisfied: umap-learn[plot] in c:\users\michael_mi\.conda\envs\219\lib\site-packages (0.5.5)
Requirement already satisfied: tqdm in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn[plot]) (4.64.1)
Requirement already satisfied: pynndescent>=0.5 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn[plot])
(0.5.11)
Requirement already satisfied: numpy>=1.17 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn[plot]) (1.19.
```

```
5)
Requirement already satisfied: scikit-learn>=0.22 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn[plot])
(0.24.2)
Requirement already satisfied: scipy>=1.3.1 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn[plot]) (1.5.
4)
Requirement already satisfied: numba>=0.51.2 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn[plot]) (0.5
3.1)
Collecting colorcet
  Downloading colorcet-3.0.1-py2.py3-none-any.whl (1.7 MB)
Collecting holoviews
  Downloading holoviews-1.14.9-py2.py3-none-any.whl (4.3 MB)
Requirement already satisfied: matplotlib in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn[plot]) (3.3.4)
Requirement already satisfied: pandas in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn[plot]) (1.1.5)
Requirement already satisfied: seaborn in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from umap-learn[plot]) (0.11.2)
Collecting datashader
  Downloading datashader-0.13.0-py2.py3-none-any.whl (15.8 MB)
Collecting bokeh
  Downloading bokeh-2.3.3.tar.gz (10.7 MB)
Collecting scikit-image
  Downloading scikit_image-0.17.2-cp36-cp36m-win_amd64.whl (11.5 MB)
Requirement already satisfied: llvmlite<0.37,>=0.36.0rc1 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from numba>=0.5
1.2->umap-learn[plot]) (0.36.0)
Requirement already satisfied: setuptools in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from numba>=0.51.2->umap-learn
[plot]) (58.0.4)
Requirement already satisfied: joblib>=0.11 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from pynndescent>=0.5->umap-l
earn[plot]) (1.1.1)
Requirement already satisfied: importlib-metadata>=4.8.1 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from pynndescent
>=0.5->umap-learn[plot]) (4.8.3)
Requirement already satisfied: typing-extensions>=3.6.4 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from importlib-me
tadata>=4.8.1->pynndescent>=0.5->umap-learn[plot]) (4.1.1)
Requirement already satisfied: zipp>=0.5 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from importlib-metadata>=4.8.1->
pynndescent>=0.5->umap-learn[plot]) (3.6.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from scikit-learn>=0.
22->umap-learn[plot]) (3.1.0)
Collecting PyYAML>=3.10
  Downloading PyYAML-6.0.1-cp36-cp36m-win_amd64.whl (153 kB)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from bokeh->umap-lear
n[plot]) (2.8.2)
Requirement already satisfied: Jinja2>=2.9 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from bokeh->umap-learn[plot])
(3.0.3)
Requirement already satisfied: pillow>=7.1.0 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from bokeh->umap-learn[plo
t]) (8.4.0)
Requirement already satisfied: packaging>=16.8 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from bokeh->umap-learn[plo
t]) (21.3)
Requirement already satisfied: tornado>=5.1 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from bokeh->umap-learn[plot])
(6.1)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from Jinja2>=2.9->bokeh->u
```

```
map-learn[plot]) (2.0.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from packaging>=1
6.8->bokeh->umap-learn[plot]) (3.1.1)
Requirement already satisfied: six>=1.5 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from python-dateutil>=2.1->bokeh-
>umap-learn[plot]) (1.16.0)
Collecting pyct>=0.4.4
  Downloading pyct-0.4.8-py2.py3-none-any.whl (15 kB)
Collecting param>=1.7.0
  Downloading param-1.13.0-py2.py3-none-any.whl (87 kB)
Collecting dask[complete]>=0.18.0
  Downloading dask-2021.3.0-py3-none-any.whl (925 kB)
Collecting datashape>=0.5.1
  Downloading datashape-0.5.2.tar.gz (76 kB)
Collecting xarray>=0.9.6
  Downloading xarray-0.16.2-py3-none-any.whl (736 kB)
Collecting fsspec>=0.6.0
  Downloading fsspec-2022.1.0-py3-none-any.whl (133 kB)
Collecting distributed>=2021.03.0
  Downloading distributed-2021.3.0-py3-none-any.whl (675 kB)
Collecting cloudpickle>=0.2.2
  Downloading cloudpickle-2.2.1-py3-none-any.whl (25 kB)
Collecting toolz>=0.8.2
  Downloading toolz-0.12.0-py3-none-any.whl (55 kB)
Collecting partd>=0.3.10
  Downloading partd-1.2.0-py3-none-any.whl (19 kB)
Collecting multipledispatch>=0.4.7
  Downloading multipledispatch-1.0.0-py3-none-any.whl (12 kB)
Collecting msgpack>=0.6.0
  Downloading msgpack-1.0.5-cp36-cp36m-win_amd64.whl (69 kB)
Collecting zict>=0.1.3
  Downloading zict-2.1.0-py3-none-any.whl (11 kB)
Collecting psutil>=5.0
  Downloading psutil-5.9.8-cp36-cp36m-win_amd64.whl (258 kB)
Collecting sortedcontainers!=2.0.0,!=2.0.1
  Downloading sortedcontainers-2.4.0-py2.py3-none-any.whl (29 kB)
Collecting tblib>=1.6.0
  Downloading tblib-1.7.0-py2.py3-none-any.whl (12 kB)
Collecting contextvars
  Downloading contextvars-2.4.tar.gz (9.6 kB)
Requirement already satisfied: click>=6.6 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from distributed>=2021.03.0->da
sk[complete]>=0.18.0->datashader->umap-learn[plot]) (8.0.4)
Requirement already satisfied: colorama in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from click>=6.6->distributed>=202
1.03.0->dask[complete]>=0.18.0->datashader->umap-learn[plot]) (0.4.5)
Requirement already satisfied: pytz>=2017.2 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from pandas->umap-learn[plo
t]) (2023.3.post1)
Collecting locket
  Downloading locket-1.0.0-py2.py3-none-any.whl (4.4 kB)
```

```
Collecting heapdict
  Downloading HeapDict-1.0.1-py3-none-any.whl (3.9 kB)
Collecting immutables>=0.9
  Downloading immutables-0.19-cp36-cp36m-win_amd64.whl (58 kB)
Collecting panel>=0.8.0
  Downloading panel-0.13.1-py2.py3-none-any.whl (15.7 MB)
Collecting pyviz-comms>=0.7.4
  Downloading pyviz_comms-2.2.1-py2.py3-none-any.whl (42 kB)
Collecting requests
  Downloading requests-2.27.1-py2.py3-none-any.whl (63 kB)
Collecting panel>=0.8.0
  Downloading panel-0.13.0-py2.py3-none-any.whl (15.6 MB)
  Downloading panel-0.12.7-py2.py3-none-any.whl (12.9 MB)
  Downloading panel-0.12.6-py2.py3-none-any.whl (12.9 MB)
  Downloading panel-0.12.5-py2.py3-none-any.whl (12.9 MB)
  Downloading panel-0.12.4-py2.py3-none-any.whl (12.9 MB)
  Downloading panel-0.12.3-py2.py3-none-any.whl (12.8 MB)
  Downloading panel-0.12.2-py2.py3-none-any.whl (12.8 MB)
  Downloading panel-0.12.1-py2.py3-none-any.whl (12.8 MB)
Requirement already satisfied: bleach in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from panel>=0.8.0->holoviews->umap-
learn[plot]) (4.1.0)
Collecting markdown
  Downloading Markdown-3.3.7-py3-none-any.whl (97 kB)
Requirement already satisfied: importlib-resources in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from tqdm->umap-learn
[plot]) (5.4.0)
Requirement already satisfied: webencodings in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from bleach->panel>=0.8.0->ho
loviews->umap-learn[plot]) (0.5.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from matplotlib->umap-le
arn[plot]) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from matplotlib->umap-learn[p
lot]) (0.11.0)
Collecting urllib3<1.27,>=1.21.1
  Downloading urllib3-1.26.18-py2.py3-none-any.whl (143 kB)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\michael_mi\.conda\envs\219\lib\site-packages (from requests->panel>=
0.8.0->holoviews->umap-learn[plot]) (2021.5.30)
Collecting idna<4,>=2.5
  Downloading idna-3.6-py3-none-any.whl (61 kB)
Collecting charset-normalizer~=2.0.0
  Downloading charset_normalizer-2.0.12-py3-none-any.whl (39 kB)
Collecting networkx>=2.0
  Downloading networkx-2.5.1-py3-none-any.whl (1.6 MB)
Collecting tifffile>=2019.7.26
  Downloading tifffile-2020.9.3-py3-none-any.whl (148 kB)
Collecting PyWavelets>=1.1.1
  Downloading PyWavelets-1.1.1-cp36-cp36m-win_amd64.whl (4.2 MB)
Collecting imageio>=2.3.0
  Downloading imageio-2.15.0-py3-none-any.whl (3.3 MB)
```

```
Collecting decorator<5,>=4.3
  Downloading decorator-4.4.2-py2.py3-none-any.whl (9.2 kB)
Building wheels for collected packages: bokeh, datashape, contextvars
  Building wheel for bokeh (setup.py): started
  Building wheel for bokeh (setup.py): finished with status 'done'
  Created wheel for bokeh: filename=bokeh-2.3.3-py3-none-any.whl size=11342794 sha256=84afc0c4232244aa18b268d62dde7521f0b17d4cf8a2a
a006b23e7043b78a1fd
  Stored in directory: c:\users\michael_mi\appdata\local\pip\cache\wheels\8b\59\97\257265b741bab184e0cc8f5676309cb1fe6fbda22011bbb3
ff
  Building wheel for datashape (setup.py): started
  Building wheel for datashape (setup.py): finished with status 'done'
  Created wheel for datashape: filename=datashape-0.5.2-py3-none-any.whl size=59453 sha256=143467c07b323a95e4ef548b8f008268c45042a7
a89ae0aec7aba512d2524ada
  Stored in directory: c:\users\michael_mi\appdata\local\pip\cache\wheels\81\61\fc\7d268954f6907b2a547c7895012769cde53af58f1aaf95a5
4c
  Building wheel for contextvars (setup.py): started
  Building wheel for contextvars (setup.py): finished with status 'done'
  Created wheel for contextvars: filename=contextvars-2.4-py3-none-any.whl size=7681 sha256=fed357c9403f3bb6aef6e8673bb151bf01356ec
b89b5228e3621aef5df5c2708
  Stored in directory: c:\users\michael_mi\appdata\local\pip\cache\wheels\41\11\53\911724983aa48deb94792432e14e518447212dd6c5477d49
d3
Successfully built bokeh datashape contextvars
Installing collected packages: PyYAML, immutables, heapdict, zict, urllib3, toolz, tblib, sortedcontainers, psutil, param, msgpack,
locket, idna, dask, contextvars, cloudpickle, charset-normalizer, requests, pyviz-comms, pyct, partd, multipledispatch, markdown, f
sspec, distributed, decorator, bokeh, xarray, tifffile, PyWavelets, panel, networkx, imageio, datashape, colorcet, scikit-image, ho
loviews, datashader
  Attempting uninstall: decorator
    Found existing installation: decorator 5.1.1
    Uninstalling decorator-5.1.1:
      Successfully uninstalled decorator-5.1.1
Successfully installed PyWavelets-1.1.1 PyYAML-6.0.1 bokeh-2.3.3 charset-normalizer-2.0.12 cloudpickle-2.2.1 colorcet-3.0.1 context
vars-2.4 dask-2021.3.0 datashader-0.13.0 datashape-0.5.2 decorator-4.4.2 distributed-2021.3.0 fsspec-2022.1.0 heapdict-1.0.1 holovi
ews-1.14.9 idna-3.6 imageio-2.15.0 immutables-0.19 locket-1.0.0 markdown-3.3.7 msgpack-1.0.5 multipledispatch-1.0.0 networkx-2.5.1
panel-0.12.1 param-1.13.0 partd-1.2.0 psutil-5.9.8 pyct-0.4.8 pyviz-comms-2.2.1 requests-2.27.1 scikit-image-0.17.2 sortedcontainer
s-2.4.0 tblib-1.7.0 tifffile-2020.9.3 toolz-0.12.0 urllib3-1.26.18 xarray-0.16.2 zict-2.1.0
```

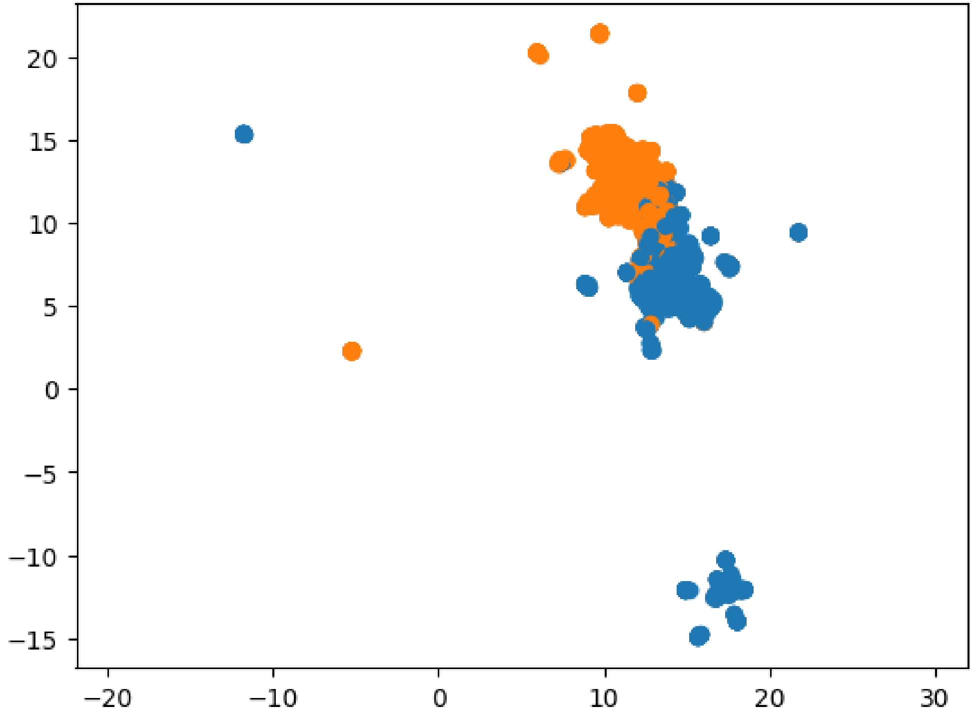
In [ ]:

```
import umap.umap_ as umap
import umap.plot
import seaborn as sns

umap_model = umap.UMAP(random_state=42)
labels = train['binary_root_label']

def apply_umap_projection(data, labels):
    umap_model = umap.UMAP(random_state=42)
    umap_result = umap_model.fit_transform(data)
```

```
    plt.scatter(
        umap_result[:, 0],
        umap_result[:, 1],
        c=[sns.color_palette()[x] for x in labels])
    plt.gca().set_aspect('equal', 'datalim')
    plt.title('UMAP Projection of the Normalized GLoVE-based Embeddings', fontsize=24)
    plt.show()

# Apply UMAP on the GLoVE training data (300 dimensions)
apply_umap_projection(X_train[300], train['binary_root_label'])
```

```
c:\Users\Michael_Mi\.conda\envs\219-ass01\lib\site-packages\umap\umap_.py:1943: UserWarning: n_jobs value -1 overridden to 1 by set
ting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use no seed for parallelism.")
```

UMAP Projection of the Normalized GLoVE-based Embeddings

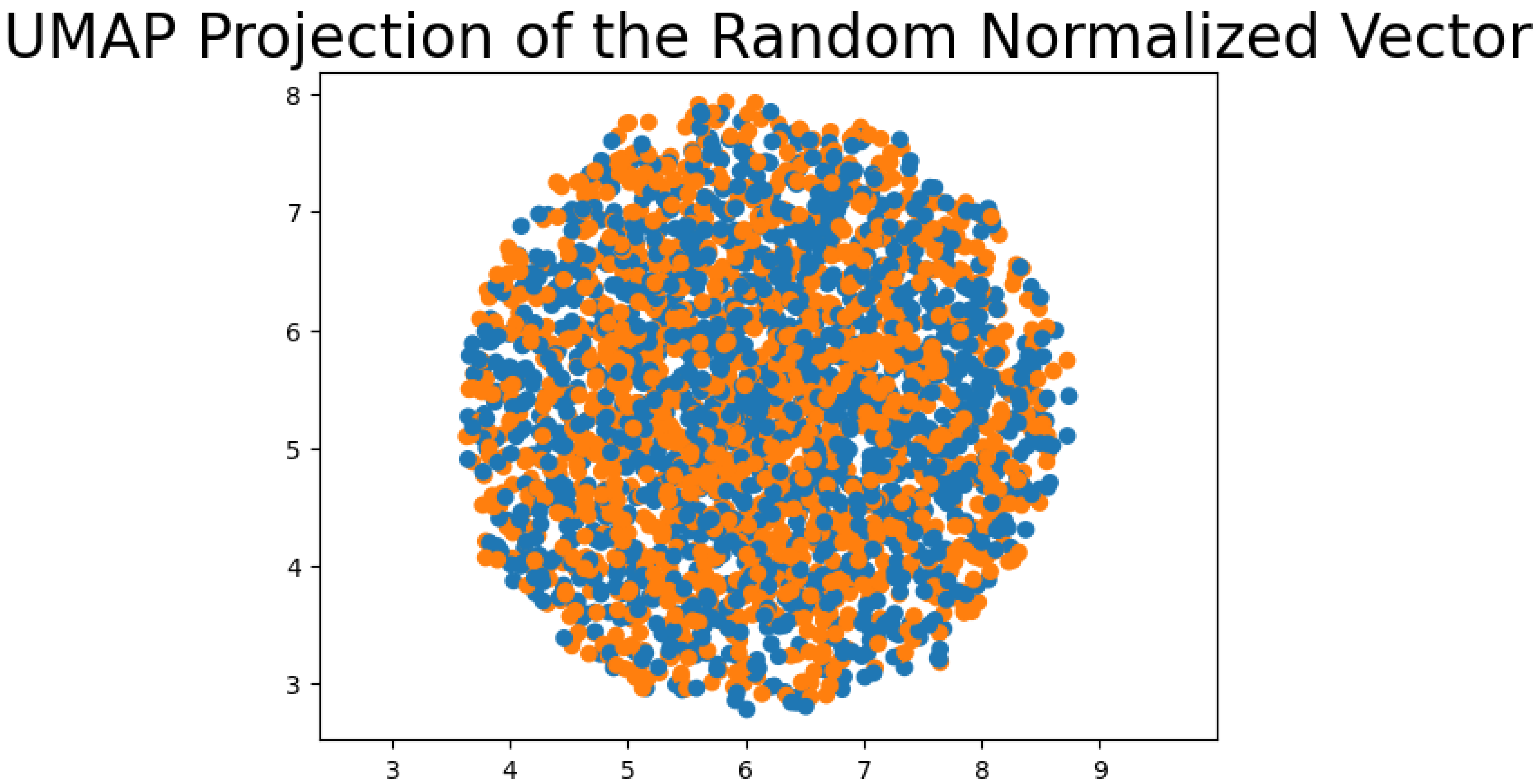


In [ ]:

```
# Generate random points and normalize them
def generate_and_normalize_points(num_samples, vector_size):
    random_samples = np.random.normal(loc=0.0, scale=1.0, size=(num_samples, vector_size))
```

```python
    normalized_samples = [sample / np.linalg.norm(sample) for sample in random_samples]
    return normalized_samples

# Apply UMAP on the random normalized vectors
def apply_umap_on_normalized_vectors(normalized_vectors, labels):
    embedding_result = umap_model.fit_transform(normalized_vectors)

    # Plot
    plt.scatter(
        embedding_result[:, 0],
        embedding_result[:, 1],
        c=[sns.color_palette()[x] for x in labels])
    plt.gca().set_aspect('equal', 'datalim')
    plt.title('UMAP Projection of the Random Normalized Vector', fontsize=24)
    plt.show()

# Generate and normalize random points
normalized_random_vectors  = generate_and_normalize_points(2780, 300)

# Apply UMAP on the random normalized vectors
apply_umap_on_normalized_vectors(normalized_random_vectors, labels)
```

```
c:\Users\Michael_Mi\.conda\envs\219-ass01\lib\site-packages\umap\umap_.py:1943: UserWarning: n_jobs value -1 overridden to 1 by set
ting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use no seed for parallelism.")
```



Q13 Text Answer:

When comparing the two virtualizations, the plot of normalized GLOVE shows clusters, however, the plot with random vectors lacks clusters, and does not show any clear meaningful distribution. This means that GLOVE is effective when capture semantic similar contents, and this is the cause of the distinct clusters in the graph. On the other hand This emphasized the importance of meaningful embedding, and this also underscored the GLOVE's ability to representing content in document classification.