ECE 219 Project 1

Group Members: Zan Xie (UID: 205364923), Joseph Gong (UID: 606073799), Anuk Fernando (UID: 805423707)

mount google drive and load the data file

input file: Project1-ClassificationDataset.csv

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
df = pd.read_csv('/content/drive/Shareddrives/ECE219/Project1-Classificatior
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
ll drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
print(df['full_text'])
```

```
0       'Personalize Your NBA App Experience for the '...
1       'Mike Will attends the Pre-GRAMMY Gala and GRA...
2       'The Golden State Warriors are struggling to f...
3       'On Nov. 28, the NBA and Nike will collaborate...
4       'The NBA announced additions and innovations t...
                              ...
3471    'The Virginia Department of Forestry continues...
3472    'State Alabama Alaska Arizona Arkansas Califor...
3473    'Chengdu showcases technological strength at h...
3474    'Bluefield, WV (24701)\n\nToday\n\nPartly clou...
3475    'The Search for Extraterrestrial Intelligence ...
Name: full_text, Length: 3476, dtype: object
```

Getting Farmiliar with the Dataset

Q1

(1) Overview: There are 3476 rows and 8 columns in the dataset

(2) Histogram: plots showned below.

(3)Interpret Plots: The majority have less than 10 thousand Alpha-numeric characters in a news text, and more than half of the news have characters less than 3 thousand. In the dataset, the number of news are evenly distribute to 10 leaf catagories. Regarding of root catagories, the number is evenly distributed. Expect that drought catagory has less than others.

In [ ]:
```
display(df.head())
print("rows: ", df.shape[0], " and columns: ", df.shape[1])
```

	full_text	summary	keywords	publish_date	authors	
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/ne
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/ent
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/
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/w
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.co

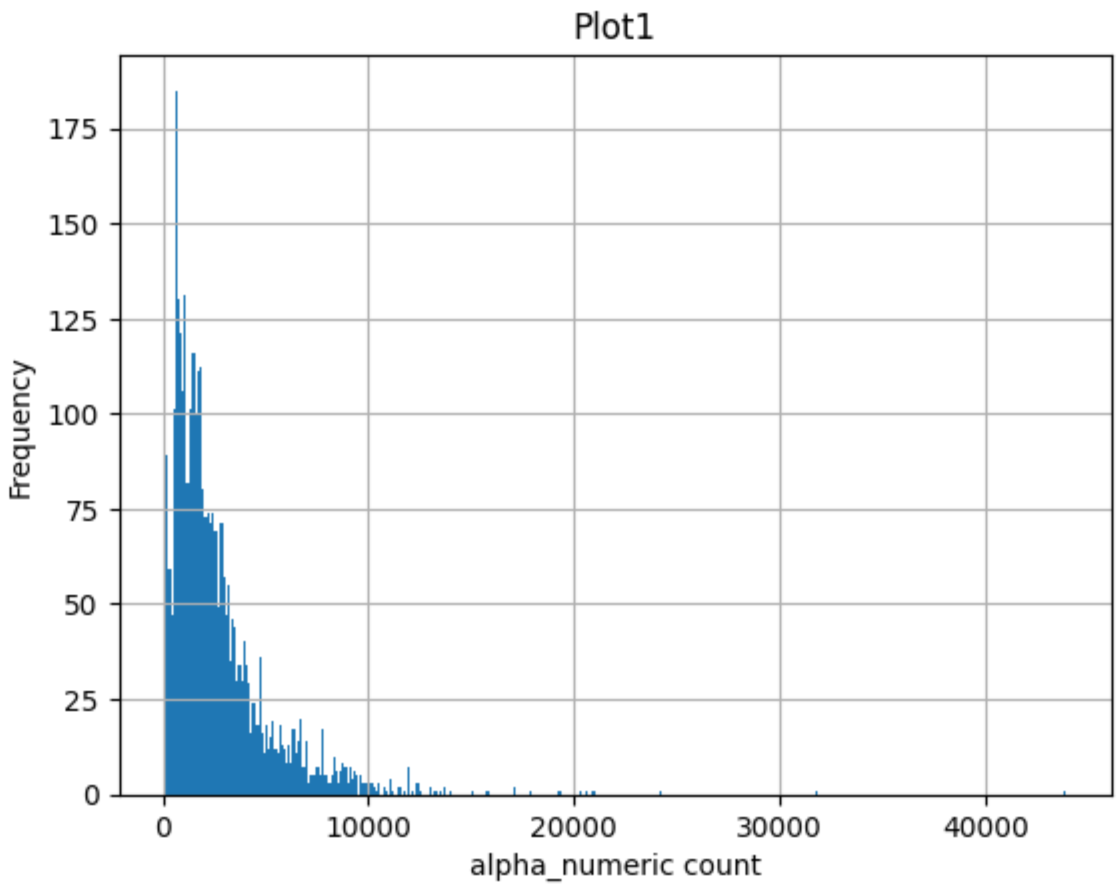
```
row:  3476  and columns:  8
```

In [ ]:

```
# number of alpha-numeric plot
num_char = df['full_text'].str.len() - df['full_text'].str.count('\s')

# plot histogram with bins = 400 (roughly interval = 100)
alph_plot = num_char.hist(bins=400)
alph_plot.set_title("Plot1")
alph_plot.set_xlabel("alpha_numeric count")
alph_plot.set_ylabel("Frequency")
```

Out[ ]:

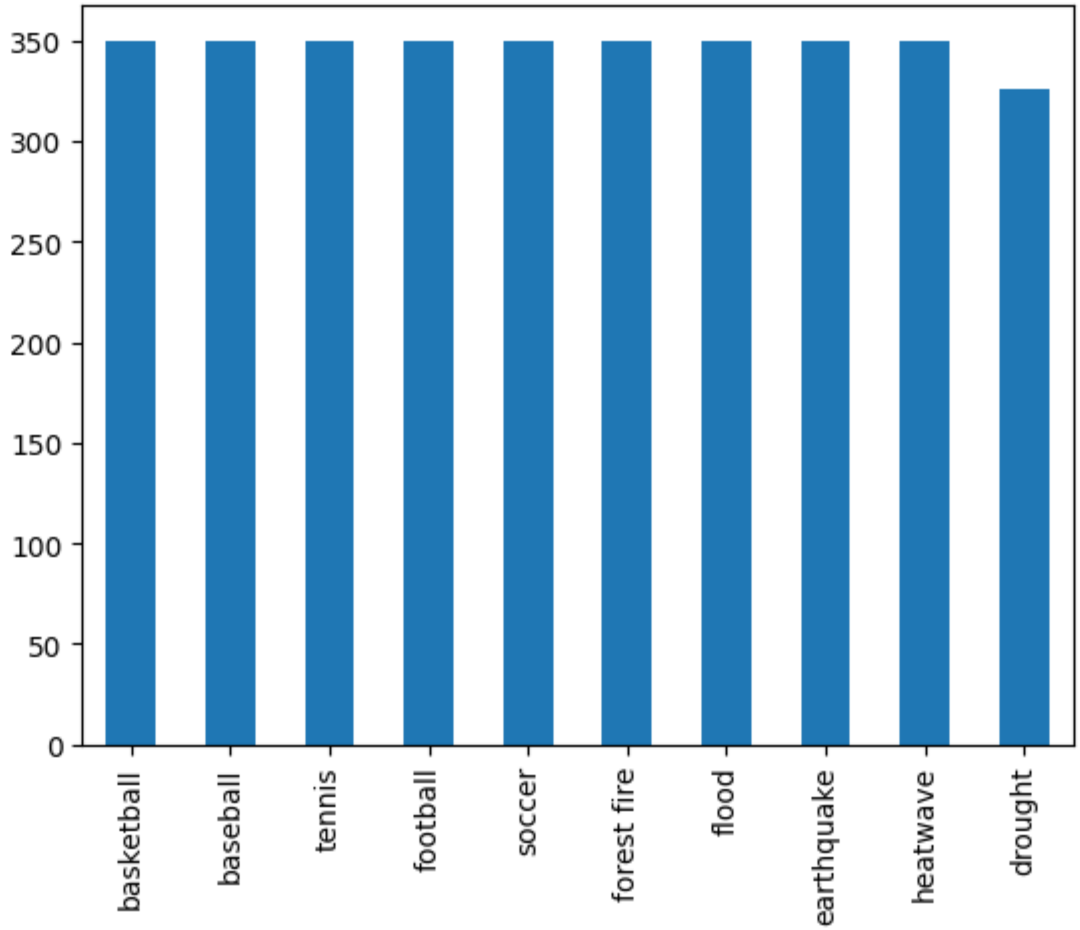
```
Text(0, 0.5, 'Frequency')
```



In [ ]:
```
print('Minimum: ', num_char.min(), '\n Maximum: ', num_char.max())
```

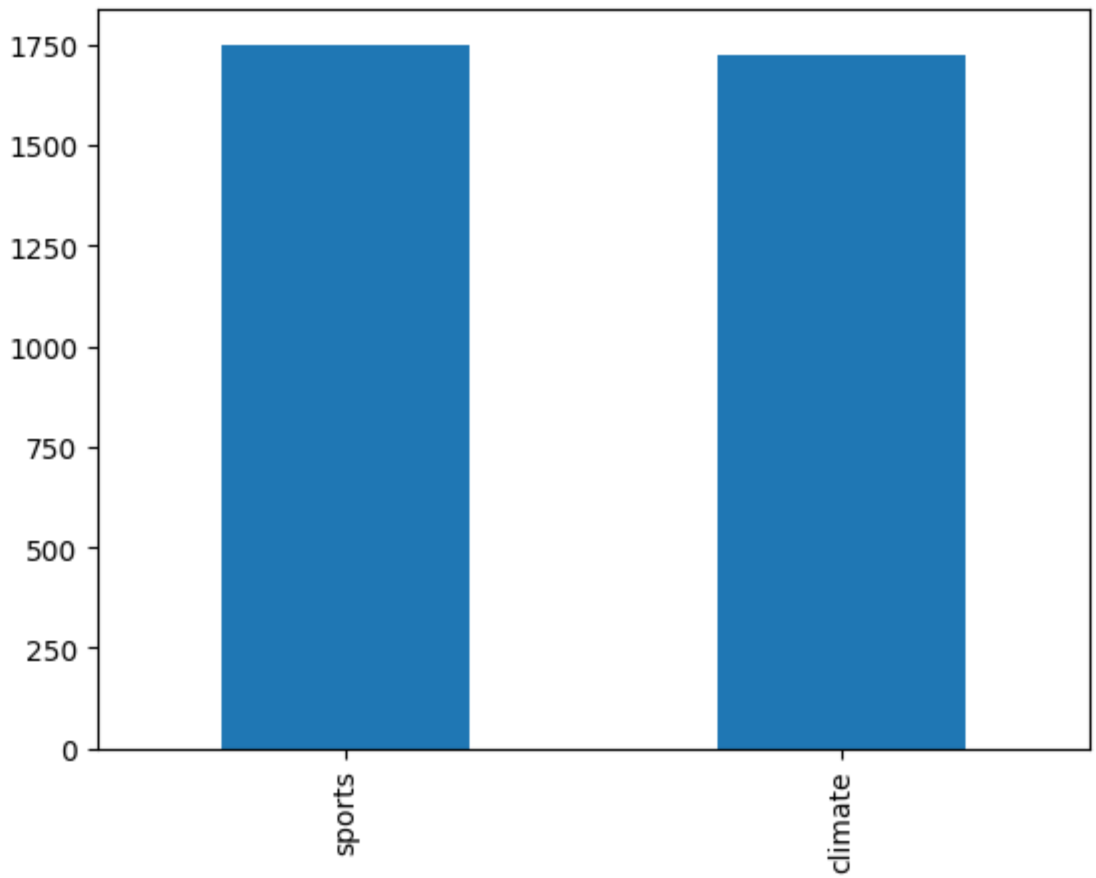
```
Minimum:  46
 Maximum:  43922
```

In [ ]:
```
# leaf_label plot
import matplotlib.pyplot as plt
df['leaf_label'].value_counts().plot.bar()
plt.show()
```



In [ ]:
```
# root_label plot
import matplotlib.pyplot as plt
df['root_label'].value_counts().plot.bar()
plt.show()
```

Loading [MathJax]/extensions/Safe.js



Binary Classification

Q2 Splitting the entire dataset into training and testing data

The size of the training set is 2780.

The size of the testing set is 696.

In [ ]:
```
# random seed for consistency
import numpy as np
import random
np.random.seed(42)
random.seed(42)
```

In [ ]:
```
# seperate training and testing sets
from sklearn.model_selection import train_test_split
train, test = train_test_split(df[["full_text","root_label"]], test_size=0.2

print (train)
print (test)
print (train.shape[0])
print (test.shape[0])
```

Loading [MathJax]/extensions/Safe.js

```
                                              full_text root_label
2677  'While the four-day Aftershock's economic impa...    climate
1204  'CBS Essentials is created independently of th...     sports
2955  'Moderate-to-severe drought will likely contin...    climate
2266  'Colleen Flood, the longtime co-owner of The F...    climate
611   'WASHINGTON TRAFFIC MAY HAVE SAVED HIS LIFE. Y...     sports
...                                                 ...        ...
1095  '(Photo by Justin Casterline/Getty Images)\n\n...     sports
1130  'COOKEVILLE, Tenn. (WKRN) — The Golden Eagles ...     sports
1294  'FanDuel Sportsbook has launched an exclusive ...     sports
860   'Hunting stories are a Maine tradition, just l...     sports
3174  'By Lewis Jackson\n\nSYDNEY (Reuters) -Thousan...    climate

[2780 rows x 2 columns]
                                              full_text root_label
2069  'A small patch of snow on the ground in Douai,...    climate
1425  'Antonio Zago, of Brazil, puts on a jersey dur...     sports
309   'NEW YORK >> The Las Vegas Aces became the fir...     sports
2270  'Christian Abraham/Hearst Connecticut Media\n\...    climate
3037  'The City of Watertown is currently under a wa...    climate
...                                                 ...        ...
547   'Jasper, TX (75951)\n\nToday\n\nPeriods of rai...     sports
776   'The ATP Finals — the final tennis championshi...     sports
2873  'BOSTON — The regulations directing how the st...    climate
2236  'After weeks of infighting and turmoil that ha...    climate
568   'State Alabama Alaska Arizona Arkansas Califor...     sports

[696 rows x 2 columns]
2780
696
```

Q3 Feature Extraction

1. Lemmatization vs. Stemming
   In general, the advantages of stemming are that it's straightforward to implement and fast to run. The trade-off here is that the output might contain inaccuracies, although they may be irrelevant for some tasks, like text indexing.

   Instead, lemmatization provides better results by performing an analysis that depends on the word's part-of-speech and producing real, dictionary words. As a result, lemmatization is harder to implement and slower compared to stemming.

2. min_df & tfidf matrix
   When min_df=7, the tfidf matrix has 7361 columns. When min_df=5, the tfidf matrix has 9248 columns. When min_df=3, the matrix has 13077 columns. Increasing min_df will reduce the column number of the tfidf matrix since more words will be filtered out and the vocabulary will be reduced. If min_df>0 and min_df<1, it means the percentage of the total documents (i.e) min_df = 0.2, no less than 20% of the total documents.

3. The proper order of text processing: remove stop words -> remove punctuation -> remove numbers -> lemmatizing. stopwords are typically removed early in the process to focus on the more meaningful words. Punctuation removal is also usually done early to ensure that words are correctly tokenized.umbers may not have a lemmatized form, and their presence might not add much semantic meaning to the text.

4. Train_set TF-IDF matrix = (2780 x 13077)
   Test_set TF-IDF matrix = (696 x 13077)

In [ ]:
```
import re
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)
  texter = re.sub('\r', ' ',texter)
  texter = re.sub(r'\d+', '', texter) # exclude numbers
  texter = re.sub('[^a-zA-Z0-9\n]', ' ', texter) # Replace characters A-Za-z
  texter = re.sub('\s+',' ', texter) # Removing whitespace and newlines
  texter = texter.lower() # convert to lower case
  clean = re.compile('<.*?>')
  texter = texter.encode('ascii', 'ignore').decode('ascii')
  texter = re.sub(clean, '', texter)
  if texter == "":
      texter = ""
  return texter
```

In [ ]:
```
# clean the document text
train_clean = train['full_text'].apply(clean)
test_clean = test['full_text'].apply(clean)
```

In [ ]:
```
#lemmatization
import nltk
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
lemmatizer = nltk.stem.WordNetLemmatizer()
wordnet_lemmatizer = WordNetLemmatizer()
#stop = stopwords.words('english')


def nltk_to_wordnet_tag(nltk_tag):
```

```
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

def lemmatize_sentence(sentence):
    #tokenize the sentence and find the POS tag for each token
    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
    #tuple of (token, wordnet_tag)
    wordnet_tagged = map(lambda x: (x[0], nltk_tag_to_wordnet_tag(x[1])), nl
    lemmatized_sentence = []
    for word, tag in wordnet_tagged:
        if tag is None:
            #if there is no available tag, append the token as is
            lemmatized_sentence.append(word)
        else:
            #else use the tag to lemmatize the token
            lemmatized_sentence.append(lemmatizer.lemmatize(word, tag))
    return " ".join(lemmatized_sentence)


train_clean_lemm = train_clean.apply(lemmatize_sentence)
test_clean_lemm = test_clean.apply(lemmatize_sentence)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

In [ ]:

```
# min_df=7
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=7, stop_words='english')

X_train = vectorizer.fit_transform(train_clean_lemm)
X_test = vectorizer.transform(test_clean_lemm)
print(X_train.shape)
print(X_test.shape)
```

```
(2780, 7361)
(696, 7361)
```

In [ ]:

```
# min_df=5
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=5, stop_words='english')

X_train = vectorizer.fit_transform(train_clean_lemm)
torizer.transform(test_clean_lemm)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(2780, 9248)
(696, 9248)
```

In [ ]:
```
# min_df=0.2 (20% of the documents)
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=0.2, stop_words='english')

X_train = vectorizer.fit_transform(train_clean_lemm)
X_test = vectorizer.transform(test_clean_lemm)
print(X_train.shape)
print(X_test.shape)
```

```
(2780, 54)
(696, 54)
```

In [ ]:
```
# count vectorization, min_df = 3
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=3, stop_words='english')

X_train = vectorizer.fit_transform(train_clean_lemm)
X_test = vectorizer.transform(test_clean_lemm)
print(X_train.shape)
print(X_test.shape)
```

```
(2780, 13077)
(696, 13077)
```

In [ ]:
```
# IT_IDF transform
from sklearn.feature_extraction.text import TfidfTransformer
tfidf = TfidfTransformer()

X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

print(X_train_tfidf.shape)
print(X_test_tfidf.shape)
```

```
(2780, 13077)
(696, 13077)
```

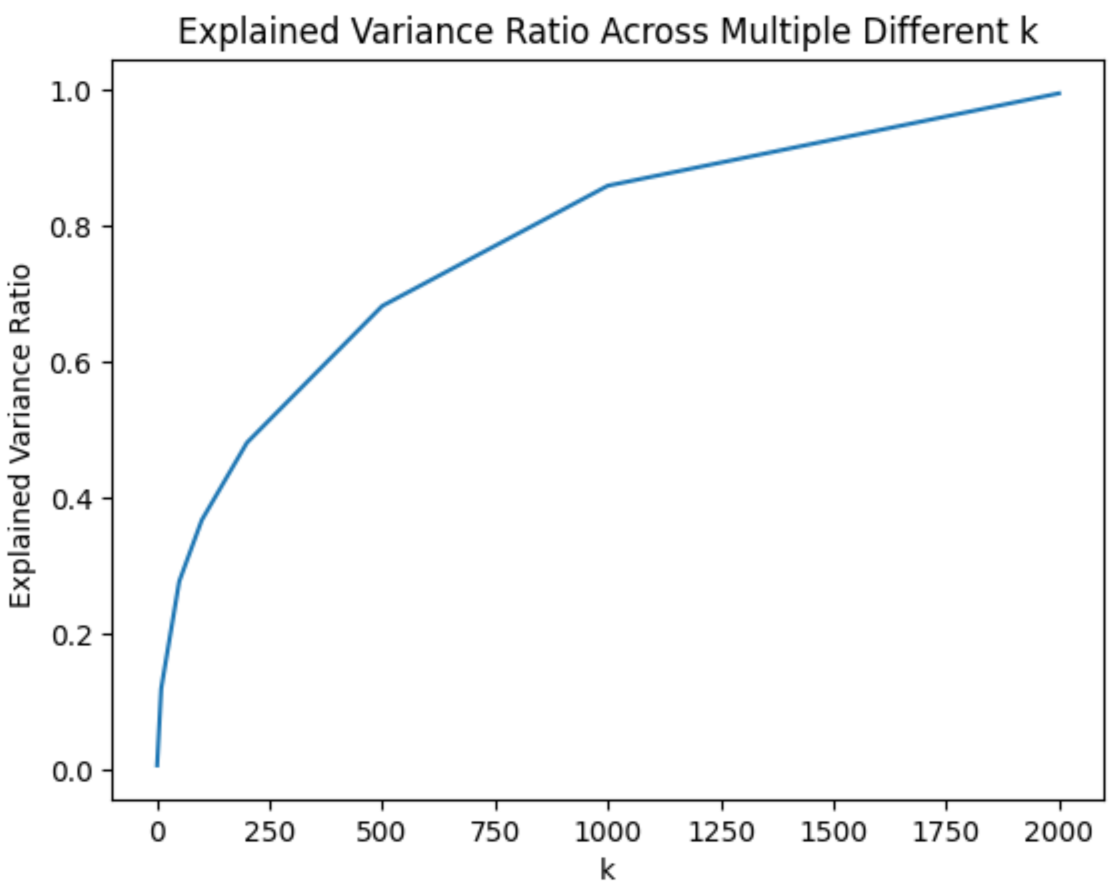
Q4 Dimensionality Reduction

(1) The plot looks like a upper curve increasing rapidly at first and gradually converging to a certain value. It implies that the explained variance ratio will be saturated as k increases.
(2) MSE for NMF is larger than the MSE for LSI. This reason is probably that the NMF is a more approximation-like methodology for dimension reduction, as it is indicated in the text that X is approximately equal to W*H.

Latent Semantic Indexing (LSI)

In [ ]:
```python
# Define the number of components k
k=[1, 10, 50, 100, 200, 500, 1000, 2000]

# Create SVD object
from sklearn.decomposition import TruncatedSVD
variance_set = []
for x in k:
  svd = TruncatedSVD(n_components=x, n_iter=10, random_state=42)
  X_train_svd = svd.fit_transform(X_train_tfidf)
  X_test_svd = svd.transform(X_test_tfidf)
  variance_ratio = svd.explained_variance_ratio_.sum()
  variance_set.append(variance_ratio)

plt.plot(k,variance_set)

plt.title('Explained Variance Ratio Across Multiple Different k')
plt.xlabel('k')
plt.ylabel('Explained Variance Ratio ')
```

Out[ ]:
```
Text(0, 0.5, 'Explained Variance Ratio ')
```



In [ ]:
```python
# LSI model with k=50
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=50, n_iter=10, random_state=42)
X_train_svd = svd.fit_transform(X_train_tfidf)
X_test_svd = svd.transform(X_test_tfidf)

# ... back to tfidf and compare with the original tfidf
```

Loading [MathJax]/extensions/Safe.js

```
X_train_svd_to_tfidf = svd.inverse_transform(X_train_svd)

from sklearn.metrics import mean_squared_error
lsi_mse = mean_squared_error(X_train_tfidf.toarray(), X_train_svd_to_tfidf)

print("MSE for LSI:", lsi_mse)
```

```
MSE for LSI: 5.348159866007923e-05
```

Non-negative Matrix Factorization (NMF)

In [ ]:
```
# initialize the model
from sklearn.decomposition import NMF
model = NMF(n_components=50, init='random', random_state=42)
W_train_nmf = model.fit_transform(X_train_tfidf)
W_test_nmf = model.transform(X_test_tfidf)
H = model.components_

# vector multiplication WH and compare with original tfidf
V_train_nmf = W_train_nmf @ H

from sklearn.metrics import mean_squared_error
nmf_mse = mean_squared_error(X_train_tfidf.toarray(), V_train_nmf)
print("MSE for NMF:", nmf_mse)
```

```
MSE for NMF: 5.4346560650741494e-05
```

Q5 Classification Algorithms

- Comparing gamma=0.00001 and gamma=1000, the latter case performs better in all ressults. The case of gamma=100000 has a better performance than gamma=1000 in accuracy, precision, and f1-score.
- The soft margin does not perform as expected. Looking at the confusion matrix, the second column equals to 0, meaning that no negative prediction. The performance of an SVM is sensitive to its hyperparameters, such as the regularization parameter C. the C=0.00001 used here is too far away from the optimal C=100 we found later.
- THe ROC reflects the performance of soft margin SVM. The ROC curve is created by plotting the true positive rate (Sensitivity) against the false positive rate (1 - Specificity) at various threshold settings.

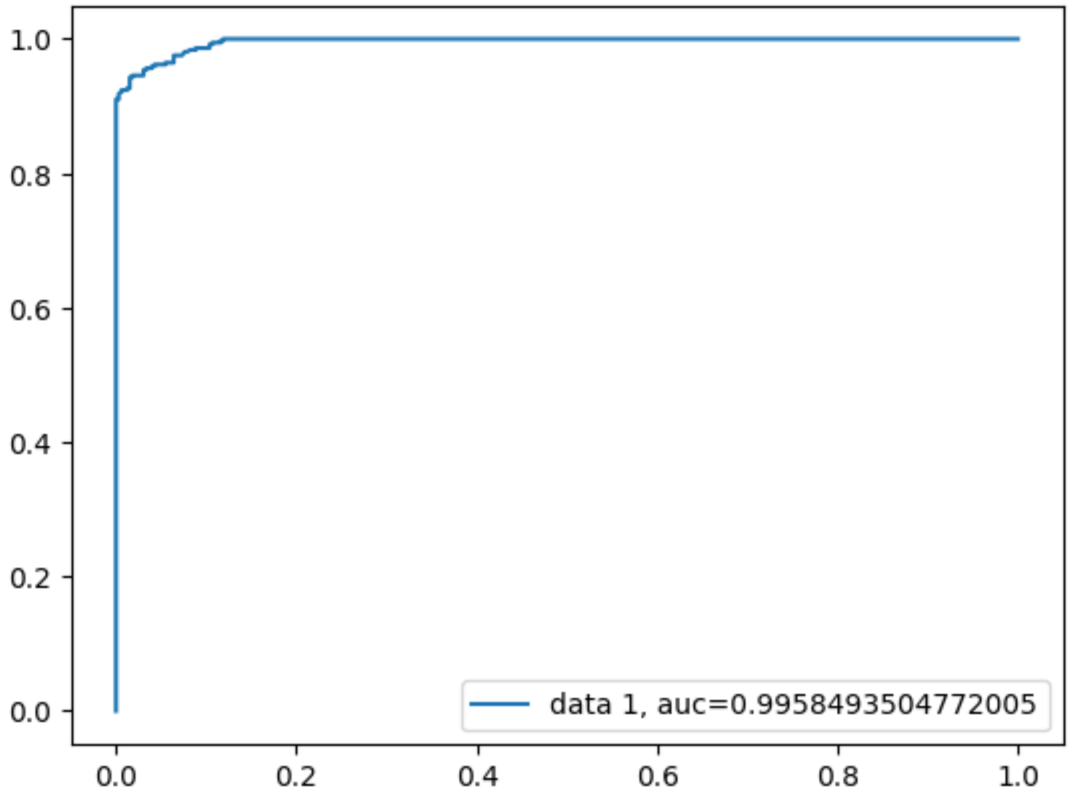
In [ ]:
```
# labeling and training model
from sklearn.svm import SVC
# label convertion
Y_train_label= train["root_label"]
Y_test_label = test["root_label"]

#convert categorical label to 0 and 1.
Y_train_label[Y_train_label  == 'sports'] = 1
Y_train_label[Y_train_label  == 'climate'] = 0
Y_train_label= Y_train_label.astype(int)

Y_test_label  == 'sports'] = 1
```

Loading [MathJax]/extensions/Safe.js

```
Y_test_label[Y_test_label  == 'climate'] = 0
Y_test_label = Y_test_label.astype(int)

# Create an SVC model
svm_1000 = SVC(C=1000, kernel='linear', probability=True)
svm_00001 = SVC(C=0.0001, kernel='linear', probability=True)
svm_100000 = SVC(C=100000, kernel='linear', probability=True)

# Train the model
svm_1000.fit(X_train_svd, Y_train_label)
svm_00001.fit(X_train_svd, Y_train_label)
svm_100000.fit(X_train_svd, Y_train_label)

# Make predictions on the test set
y_pred_1000 = svm_1000.predict(X_test_svd)
y_pred_00001 = svm_00001.predict(X_test_svd)
y_pred_100000 = svm_100000.predict(X_test_svd)
```

In [ ]:
```
# Evaluation function
from sklearn import metrics
import matplotlib.pyplot as plt
def eval_model(model, X_test_svd, Y_test_label, y_pred, roc_idx):
  #svc_disp = metrics.RocCurveDisplay.from_estimator(model, X_test_svd, Y_te
  cm = metrics.confusion_matrix(Y_test_label, y_pred) # Confusion matrix
  acc = metrics.accuracy_score(Y_test_label, y_pred) # Accuracy
  recall = metrics.recall_score(Y_test_label, y_pred) # Recall
  precision = metrics.precision_score(Y_test_label, y_pred) # Precision
  f1 = metrics.f1_score(Y_test_label, y_pred) # F-1 score

  # AUC curve with more decimal digits
  if (roc_idx):
    y_pred_proba = model.predict_proba(X_test_svd)[::,1]
    fpr, tpr, _ = metrics.roc_curve(Y_test_label,  y_pred_proba)
    auc = metrics.roc_auc_score(Y_test_label, y_pred_proba)
    plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
    plt.legend(loc=4)
    plt.show()

  #plt.show()
  print(cm)
  print("Accuracy:", acc)
  print("Recall:", recall)
  print("Precision:", precision)
  print("F-1 score:", f1)
```

In [ ]:
```
# gamma = 1000
eval_model(svm_1000, X_test_svd, Y_test_label, y_pred_1000, 1)
```



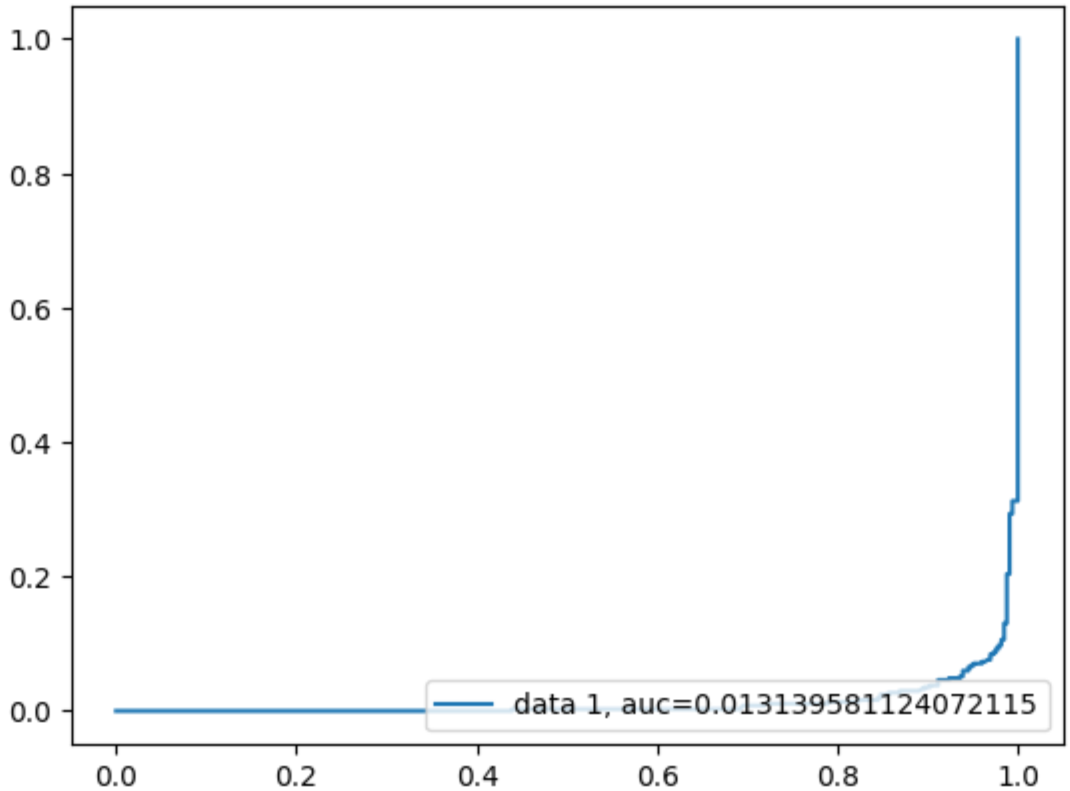

```
[[317  11]
 [ 17 351]]
Accuracy: 0.9597701149425287
Recall: 0.9538043478260869
Precision: 0.9696132596685083
F-1 score: 0.9616438356164384
```

In [ ]:

```
# gamma = 0.00001
eval_model(svm_00001, X_test_svd, Y_test_label, y_pred_00001, 1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:
1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
due to no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Loading [MathJax]/extensions/Safe.js

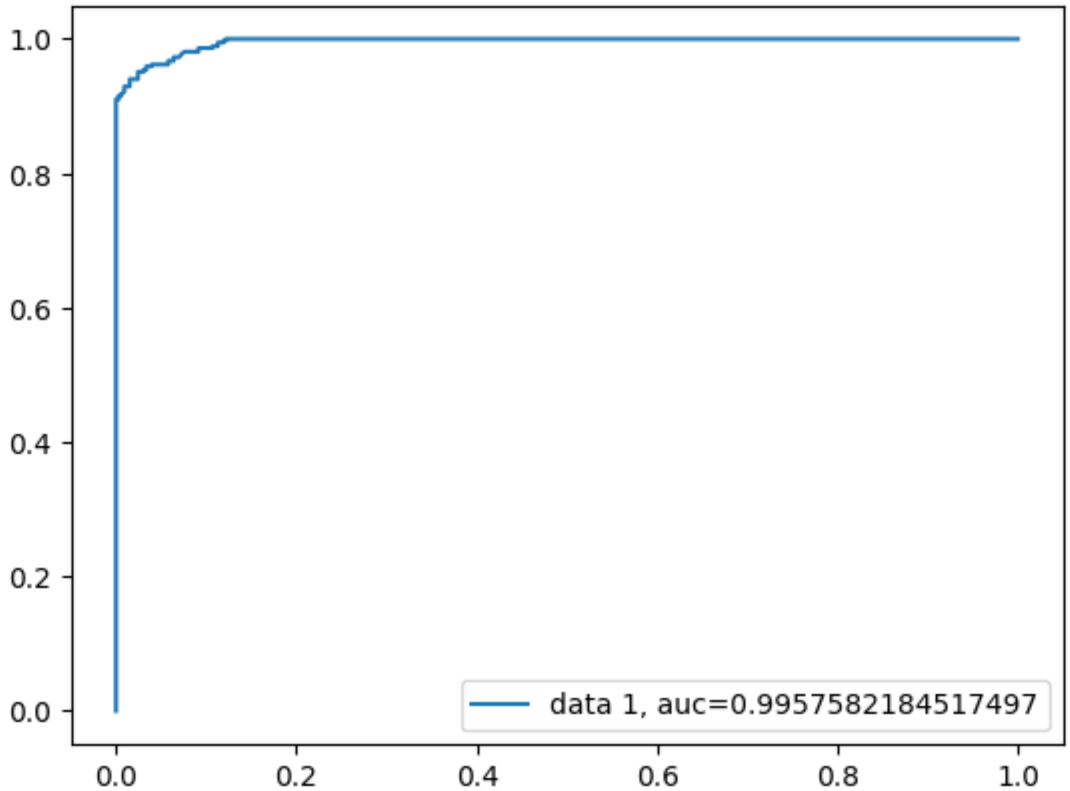



```
[[328   0]
 [368   0]]
Accuracy: 0.47126436781609193
Recall: 0.0
Precision: 0.0
F-1 score: 0.0
```

In [ ]:

```
# gamma = 100000
eval_model(svm_100000, X_test_svd, Y_test_label, y_pred_100000, 1)
```

Loading [MathJax]/extensions/Safe.js




```
[[318  10]
 [ 17 351]]
Accuracy: 0.9612068965517241
Recall: 0.9538043478260869
Precision: 0.9722991689750693
F-1 score: 0.9629629629629629
```

In [ ]:

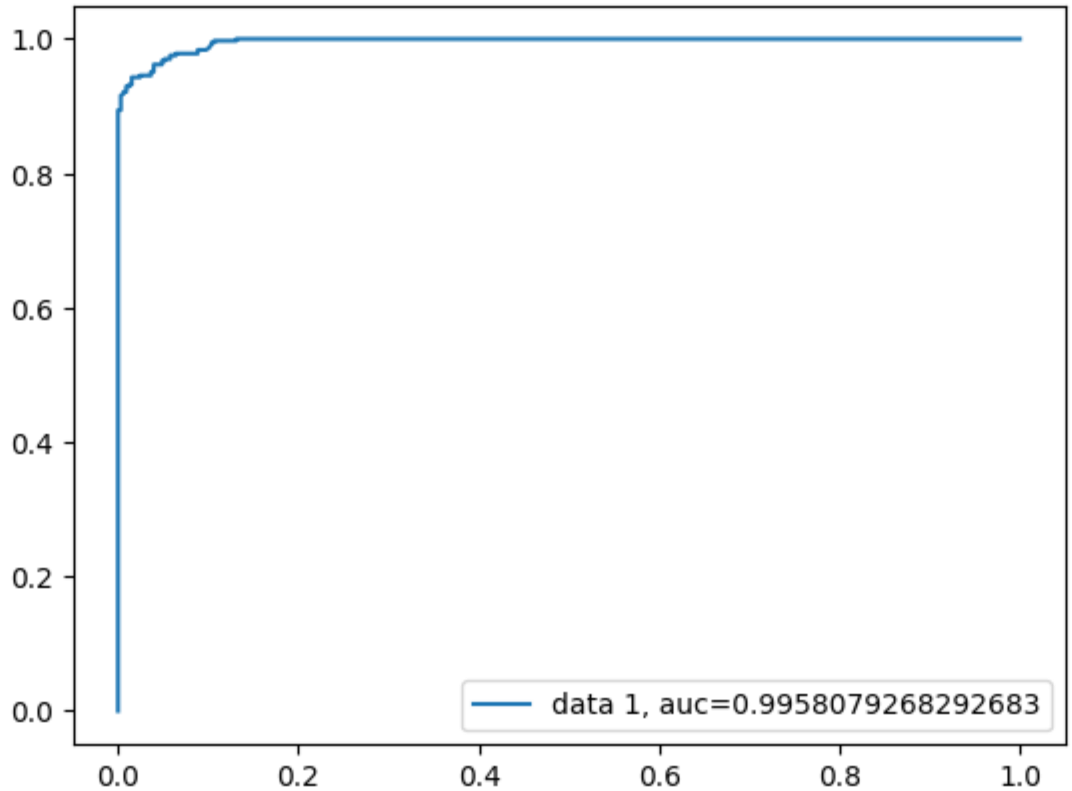
```
# Gridsearch the optimal gamma
from sklearn.model_selection import GridSearchCV
parameters = {'C':[10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4, 10
svm_opt = SVC (kernel='linear', probability=True)
grid_svm = GridSearchCV(svm_opt, parameters, cv=5, scoring='accuracy')
grid_svm.fit(X_train_svd, Y_train_label)

gamma_opt = grid_svm.best_params_['C']
print("optimal gamma :", gamma_opt )
```

```
optimal gamma : 100
```

In [ ]:

```
# opt_gamma = 100
svm_100 = SVC(C=100, kernel='linear', probability=True)
svm_100.fit(X_train_svd, Y_train_label)
y_pred_100 = svm_100.predict(X_test_svd)

eval_model(svm_100, X_test_svd, Y_test_label, y_pred_100, 1)
```

Loading [MathJax]/extensions/Safe.js




```
[[315  13]
 [ 17 351]]
Accuracy: 0.9568965517241379
Recall: 0.9538043478260869
Precision: 0.9642857142857143
F-1 score: 0.9590163934426229
```

Q6 Logistic Regression

(1)

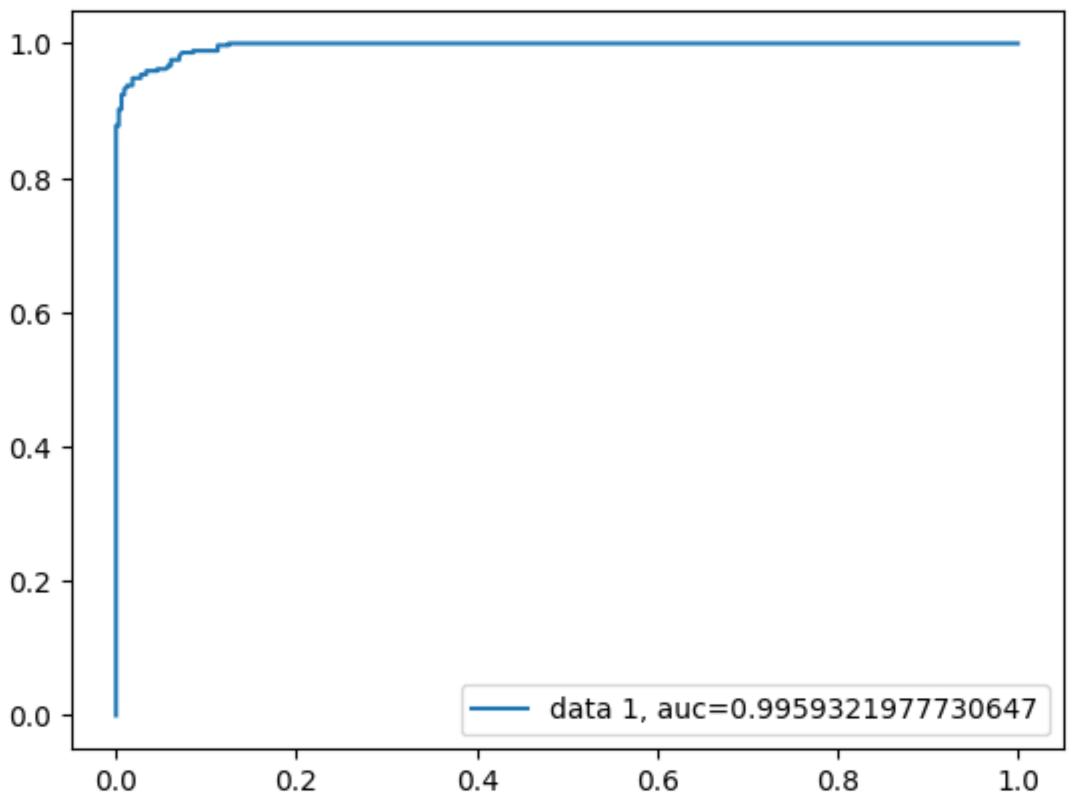
- The model becomes more restrictive as regularization strength increases, resulting in smaller learned coefficients and larger test errors
- The L1 model tends to assign significant weights to many features. Coefficients can therefore take larger values. As L2 has high regulation encourages the model to use simpler coefficients, effectively shrinking less informative features towards zero.
- L1 is useful for feature selection, especially when dealing with high-dimensional data where many features may be irrelevant. L2 is effective for handling multicollinearity among features by distributing weights more evenly.

(2)

- Logistic Regression: The decision boundary is influenced by the probability estimates assigned to instances. It smoothly transitions across the boundary.
- Linear SVM: The decision boundary is determined by the support vectors that lie closest to the decision boundary, and it focuses on maximizing the margin between classes.

- Their performance is different because Losgistic output is directly interpretable as probabilities while SVM focus is on creating a margin between classes. It's the way they dealing with the data different. The performance difference can also depend on the specific characteristics of the dataset.
- The statistically significance depends on the specific application as these two have their own specialties.

In [ ]:
```
# non-regulated logistic
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(penalty=None, max_iter=100000)

log_reg.fit(X_train_svd, Y_train_label)

y_pred_log = log_reg.predict(X_test_svd)

eval_model(log_reg, X_test_svd, Y_test_label, y_pred_log, 1)
```




```
[[314  14]
 [ 15 353]]
Accuracy: 0.9583333333333334
Recall: 0.9592391304347826
Precision: 0.9618528610354223
F-1 score: 0.9605442176870749
```

In [ ]:
```
# Gridsearch the optimal regularization strength
from sklearn.model_selection import GridSearchCV
parameters = {'C':[10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2,
log_reg_l1 = LogisticRegression(penalty='l1', solver='liblinear', max_iter=1
log_reg_l2 = LogisticRegression(penalty='l2', solver='liblinear', max_iter=1
```

Loading [MathJax]/extensions/Safe.js

```
grid_l1 = GridSearchCV(log_reg_l1, parameters, cv=5, scoring='accuracy')
grid_l2 = GridSearchCV(log_reg_l2, parameters, cv=5, scoring='accuracy')

grid_l1.fit(X_train_svd, Y_train_label)
grid_l2.fit(X_train_svd, Y_train_label)

L1_opt = grid_l1.best_params_['C']
L2_opt = grid_l2.best_params_['C']

print("optimal regularization strength for L1 :", L1_opt )
print("optimal regularization strength for L2 :", L2_opt )
```

```
optimal regularization strength for L1 : 100
optimal regularization strength for L2 : 10
```

In [ ]:
```
# L1 regulation c=100
log_reg_l1_opt = LogisticRegression(C=100, penalty='l1', solver='liblinear',
log_reg_l1_opt.fit(X_train_svd, Y_train_label)
y_pred_l1_opt = log_reg_l1_opt.predict(X_test_svd)

eval_model(log_reg_l1_opt, X_test_svd, Y_test_label, y_pred_l1_opt, 0)
```

```
[[315  13]
 [ 15 353]]
Accuracy: 0.9597701149425287
Recall: 0.9592391304347826
Precision: 0.9644808743169399
F-1 score: 0.9618528610354223
```

In [ ]:
```
# L2 regulation c=10
log_reg_l2_opt = LogisticRegression(C=10, penalty='l2', solver='liblinear',
log_reg_l2_opt.fit(X_train_svd, Y_train_label)
y_pred_l2_opt = log_reg_l2_opt.predict(X_test_svd)

eval_model(log_reg_l2_opt, X_test_svd, Y_test_label, y_pred_l2_opt, 0)
```

```
[[313  15]
 [ 14 354]]
Accuracy: 0.9583333333333334
Recall: 0.9619565217391305
Precision: 0.959349593495935
F-1 score: 0.960651289009498
```

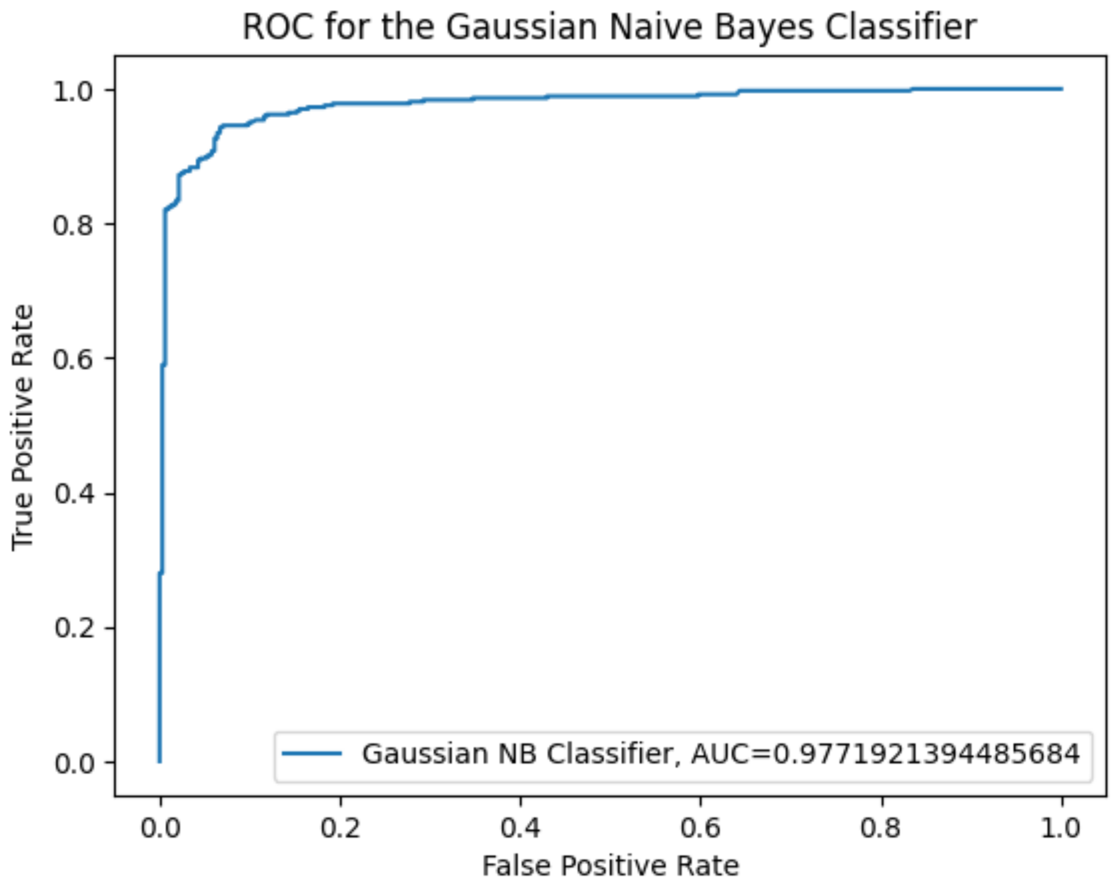
Q7 Naive Bayes Model

- The ROC curve, confusion matrix, accuracy, recall, precision, and F1-Score for the Guassian Naive Bayes model have been computed and displayed below.

In [ ]:
```
from sklearn.naive_bayes import GaussianNB

nb_classifier = GaussianNB()
nb_prediction = nb_classifier.fit(X_train_svd, Y_train_label).predict(X_test

print("NAIVE BAYES GAUSSIAN CLASSIFIER")
print("-" * 40)
```

```python
# Plotting the ROC Curve and Calculating AUC
y_pred_proba = nb_classifier.predict_proba(X_test_svd)[::,1]
fpr, tpr, _ = metrics.roc_curve(Y_test_label, y_pred_proba)
auc_value = metrics.roc_auc_score(Y_test_label, y_pred_proba)
plt.plot(fpr, tpr, label="Gaussian NB Classifier, AUC="+str(auc_value))
plt.title('ROC for the Gaussian Naive Bayes Classifier')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
print("AUC is: " + str(auc_value))
print("-" * 40)
# Computing the Confusion Matrix
confusion_matrix = metrics.confusion_matrix(Y_test_label, nb_prediction)
print("Confusion Matrix:")
print(confusion_matrix)
print("-" * 40)
# Computing Accuracy, Recall, Precision, and F1-Score
accuracy = metrics.accuracy_score(Y_test_label, nb_prediction)
print("Accuracy:", accuracy)
recall = metrics.recall_score(Y_test_label, nb_prediction)
print("Recall:", recall)
precision = metrics.precision_score(Y_test_label, nb_prediction)
print("Precision:", precision)
f1 = metrics.f1_score(Y_test_label, nb_prediction)
print("F1-Score:", f1)
```

```
NAIVE BAYES GAUSSIAN CLASSIFIER
----------------------------------------
```




```
AUC is: 0.9771921394485684
----------------------------------------
Confusion Matrix:
[[277  51]
 [ 12 356]]
----------------------------------------
Accuracy: 0.9094827586206896
Recall: 0.967391304347826
Precision: 0.8746928746928747
F1-Score: 0.9187096774193548
```

Q8 Grid Search of Parameters

According to the 5-fold grid search cross validation, here are the best performing parameter combinations, along with their corresponding average accuracy measures:

1. Stemming, min_df = 5, NMF with 80 components, Logisitc Regression with L1 regularization and C=L1_opt=100 for a **mean_test_score of 0.960791** during grid search

2. Lemmatization, min_df = 5, LSI with 80 components, Logistic Regression with L2 regularization and C=L2_opt=10 for a **mean_test_score of 0.960432** during grid search

3. Lemmatization, min_df = 3, NMF with 80 components, SVM with gamma_opt=100 for a **mean_test_score of 0.960432** during grid search
4. Lemmatization, min_df = 3, LSI with 80 components, Logistic Regression with L2 regularization and C=L2_opt=10 for a **mean_test_score of 0.960072** during grid search
5. Lemmatization, min_df = 5, LSI with 80 components, SVM with gamma_opt=100 for a **mean_test_score of 0.960072** during grid search

Then, after evaluating each of these parameter combinations on the test set, we got the following results:

Test Accuracy of 1st Best Parameter Combination: 0.9540229885057471

Test Accuracy of 2nd Best Parameter Combination: 0.9568965517241379

Test Accuracy of 3rd Best Parameter Combination: 0.9612068965517241

Test Accuracy of 4th Best Parameter Combination: 0.9612068965517241

Test Accuracy of 5th Best Parameter Combination: 0.9698275862068966

In [ ]:

```
# Implementing Stemming Functionality
train_clean = train['full_text'].apply(clean)
test_clean = test['full_text'].apply(clean)

ps = nltk.stem.PorterStemmer()

from sklearn.feature_extraction.text import CountVectorizer
analyzer = CountVectorizer().build_analyzer()

def stem_doc(doc):
    return [" ".join([ps.stem(token) for token in nltk.word_tokenize(d)]) fo

train_clean_stem = stem_doc(train_clean)
test_clean_stem = stem_doc(test_clean)

train_clean_stem = pd.Series(train_clean_stem)
test_clean_stem = pd.Series(test_clean_stem)

# train_clean_lemm
# test_clean_lemm
# are also available, from earlier
```

In [ ]:

```
from sklearn.pipeline import Pipeline
from tempfile import mkdtemp
from shutil import rmtree
from joblib import Memory

cachedir = mkdtemp()
memory = Memory(location=cachedir, verbose=10)
```

```
pipeline = Pipeline([
    ('vect', CountVectorizer(stop_words='english')),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', None),
    ('clf', None),
],
memory=memory
)

# These are the classifiers that will be compared against each other
SVM_classifier = SVC(C=gamma_opt, kernel='linear')
logregl1_classifier = LogisticRegression(C=L1_opt, penalty='l1', solver='lib
logregl2_classifier = LogisticRegression(C=L2_opt, penalty='l2', solver='lib
NB_classifier = GaussianNB()

# Comprehensive list of paramaters that will be compared during the Grid Sea
param_grid = [
    {
        "vect__min_df": [3, 5],
        'reduce_dim': [TruncatedSVD(n_iter=10, random_state=42), NMF(init='r
        'reduce_dim__n_components': [5, 30, 80],
        'clf': [SVM_classifier, logregl1_classifier, logregl2_classifier, NB
    }
]
```

In [ ]:

```
# Run the actual grid search to find optimal paramters for lemmatization (wa
grid_search_lemm = GridSearchCV(pipeline, cv=5, n_jobs=1, param_grid=param_g
grid_search_lemm.fit(train_clean_lemm, Y_train_label)
rmtree(cachedir)
```

```
None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 0.0s, 0.0min
___________________________________________________________________________
_____
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(NMF(init='random', n_components=80, random_state=42), <2780x8184 sparse matrix of type '<class 'numpy.float64'>'
        with 423104 stored elements in Compressed Sparse Row format>,
2677    0
1204    1
2955    0
2266    0
611     1
       ..
1095    1
1130    1
1294    1
860     1
3174    0
Name: root_label, Length: 2780, dtype: int64,
None, message_clsname='Pipeline', message=None)
_______________________________________________fit_transform_one - 15.4s, 0.3min
```

In [ ]:

```
# Display Sorted Results for Lemmatization
display(grid_search_lemm.best_params_)

table_of_results_lemm = pd.DataFrame(grid_search_lemm.cv_results_)
table_of_results_lemm.sort_values(by='mean_test_score', ascending=False, inp
pd.DataFrame(table_of_results_lemm)
```

```
{'clf': SVC(C=100, kernel='linear'),
 'reduce_dim': NMF(init='random', n_components=80, random_state=42),
 'reduce_dim__n_components': 80,
 'vect__min_df': 3}
```

Loading [MathJax]/extensions/Safe.js

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf	
29	0.147844	0.008951	0.131595	0.004489	LogisticRegression(C=10, max_iter=100000, solv...	Trun
10	18.785252	3.189914	0.308804	0.071334	SVC(C=100, kernel='linear')	
28	0.178389	0.022868	0.184256	0.032755	LogisticRegression(C=10, max_iter=100000, solv...	Trun
5	1.025888	0.148323	0.151268	0.036526	SVC(C=100, kernel='linear')	Trun
17	0.232577	0.018484	0.127195	0.003043	LogisticRegression(C=100, max_iter=100000, pen...	Trun
4	1.030678	0.029088	0.127692	0.006253	SVC(C=100, kernel='linear')	Trun
16	0.268253	0.050217	0.122109	0.004244	LogisticRegression(C=100, max_iter=100000, pen...	Trun
23	0.139601	0.003238	0.213741	0.011665	LogisticRegression(C=100, max_iter=100000, pen...	
22	0.145490	0.001547	0.234788	0.005641	LogisticRegression(C=100, max_iter=100000, pen...	
34	0.140899	0.004997	0.241008	0.011226	LogisticRegression(C=10, max_iter=100000, solv...	
2	0.573192	0.013372	0.125877	0.004588	SVC(C=100, kernel='linear')	Trun
11	15.423371	2.394794	0.233956	0.008429	SVC(C=100, kernel='linear')	
47	0.093896	0.004445	0.215246	0.005063	GaussianNB()	
14	0.143307	0.017145	0.124447	0.005655	LogisticRegression(C=100, max_iter=100000, pen...	Trun
3	0.646927	0.141825	0.161070	0.034412	SVC(C=100, kernel='linear')	Trun
15	0.148855	0.005018	0.125765	0.002019	LogisticRegression(C=100, max_iter=100000, pen...	Trun
46	0.102606	0.014452	0.267241	0.073987	GaussianNB()	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf	
21	0.155168	0.026788	0.167458	0.040213	LogisticRegression(C=100, max_iter=100000, pen...	
35	0.135757	0.002598	0.246941	0.038797	LogisticRegression(C=10, max_iter=100000, solv...	
26	0.131776	0.017519	0.125060	0.002390	LogisticRegression(C=10, max_iter=100000, solv...	Trun
27	0.138060	0.005151	0.120451	0.002865	LogisticRegression(C=10, max_iter=100000, solv...	Trun
20	0.145120	0.030381	0.186995	0.044648	LogisticRegression(C=100, max_iter=100000, pen...	
9	2.756527	1.008561	0.185233	0.032080	SVC(C=100, kernel='linear')	
8	3.714562	0.988188	0.203419	0.041068	SVC(C=100, kernel='linear')	
45	0.105092	0.018884	0.188888	0.039062	GaussianNB()	
44	0.093699	0.002503	0.152343	0.004424	GaussianNB()	
33	0.130578	0.002993	0.147092	0.007423	LogisticRegression(C=10, max_iter=100000, solv...	
32	0.128152	0.015853	0.150388	0.005280	LogisticRegression(C=10, max_iter=100000, solv...	
31	0.096971	0.004171	0.123749	0.003403	LogisticRegression(C=10, max_iter=100000, solv...	
13	0.124636	0.016443	0.172163	0.033876	LogisticRegression(C=100, max_iter=100000, pen...	Trun
12	0.097726	0.002495	0.127116	0.021410	LogisticRegression(C=100, max_iter=100000, pen...	Trun
1	1.098492	0.189922	0.148807	0.019633	SVC(C=100, kernel='linear')	Trun
19	0.095471	0.002986	0.120605	0.004616	LogisticRegression(C=100, max_iter=100000, pen...	
0	1.129549	0.192820	0.168237	0.047996	SVC(C=100, kernel='linear')	Trun

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf	
7	0.318373	0.042615	0.157031	0.005610	SVC(C=100, kernel='linear')	
24	0.103826	0.016024	0.117698	0.004415	LogisticRegression(C=10, max_iter=100000, solv...	Trun
18	0.105172	0.014537	0.125127	0.003998	LogisticRegression(C=100, max_iter=100000, pen...	
25	0.095207	0.005065	0.116533	0.003493	LogisticRegression(C=10, max_iter=100000, solv...	Trun
6	0.436830	0.116468	0.155256	0.004306	SVC(C=100, kernel='linear')	
41	0.094196	0.004149	0.122265	0.006833	GaussianNB()	Trun
30	0.105229	0.015945	0.126241	0.005292	LogisticRegression(C=10, max_iter=100000, solv...	
40	0.095999	0.001456	0.120989	0.006687	GaussianNB()	Trun
43	0.092994	0.002716	0.118177	0.004736	GaussianNB()	
39	0.094817	0.003191	0.118863	0.003570	GaussianNB()	Trun
38	0.096876	0.002788	0.126376	0.006994	GaussianNB()	Trun
42	0.097120	0.004881	0.122561	0.003980	GaussianNB()	
37	0.093792	0.002757	0.120117	0.008889	GaussianNB()	Trun
36	0.126026	0.018391	0.170425	0.046082	GaussianNB()	Trun

In [ ]:
```
# Display Sorted Results for Stemming
display(grid_search_stem.best_params_)

table_of_results_stem = pd.DataFrame(grid_search_stem.cv_results_)
table_of_results_stem.sort_values(by='mean_test_score', ascending=False, inp
pd.DataFrame(table_of_results_stem)
```

```
{'clf': LogisticRegression(C=100, max_iter=100000, penalty='l1', solver='li
blinear'),
 'reduce_dim': NMF(init='random', n_components=80, random_state=42),
 'reduce_dim__n_components': 80,
 'vect__min_df': 5}
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf	
23	0.129608	0.003896	0.217574	0.007317	LogisticRegression(C=100, max_iter=100000, pen...	
11	15.611570	3.026732	0.234493	0.006791	SVC(C=100, kernel='linear')	
29	0.138554	0.003039	0.122920	0.004072	LogisticRegression(C=10, max_iter=100000, solv...	Trun
16	0.207819	0.009409	0.133135	0.003350	LogisticRegression(C=100, max_iter=100000, pen...	Trun
5	0.998612	0.226948	0.142742	0.031770	SVC(C=100, kernel='linear')	Trun
28	0.143077	0.005437	0.124292	0.003198	LogisticRegression(C=10, max_iter=100000, solv...	Trun
17	0.234155	0.030208	0.128620	0.005176	LogisticRegression(C=100, max_iter=100000, pen...	Trun
4	0.955743	0.020517	0.126728	0.004936	SVC(C=100, kernel='linear')	Trun
22	0.159646	0.029562	0.307785	0.065442	LogisticRegression(C=100, max_iter=100000, pen...	
10	19.490791	2.699319	0.249703	0.011425	SVC(C=100, kernel='linear')	
3	0.610608	0.152273	0.154423	0.024108	SVC(C=100, kernel='linear')	Trun
35	0.119668	0.001741	0.217120	0.004953	LogisticRegression(C=10, max_iter=100000, solv...	
47	0.089585	0.012951	0.272838	0.061563	GaussianNB()	
2	0.568015	0.026179	0.130288	0.008653	SVC(C=100, kernel='linear')	Trun
14	0.167680	0.034785	0.183155	0.047662	LogisticRegression(C=100, max_iter=100000, pen...	Trun
21	0.134675	0.021720	0.150249	0.015910	LogisticRegression(C=100, max_iter=100000, pen...	
15	0.160439	0.033172	0.152739	0.036973	LogisticRegression(C=100, max_iter=100000, pen...	Trun

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf	
46	0.090692	0.001367	0.236652	0.009310	GaussianNB()	
27	0.125829	0.003365	0.118655	0.003924	LogisticRegression(C=10, max_iter=100000, solv...	Trun
26	0.125160	0.013996	0.119835	0.006065	LogisticRegression(C=10, max_iter=100000, solv...	Trun
20	0.139973	0.030913	0.215606	0.049286	LogisticRegression(C=100, max_iter=100000, pen...	
34	0.127751	0.002365	0.241086	0.014300	LogisticRegression(C=10, max_iter=100000, solv...	
9	3.429516	1.127476	0.194399	0.029623	SVC(C=100, kernel='linear')	
8	4.234770	0.677221	0.184612	0.029566	SVC(C=100, kernel='linear')	
32	0.114985	0.013401	0.150123	0.005989	LogisticRegression(C=10, max_iter=100000, solv...	
33	0.119382	0.003565	0.142567	0.004543	LogisticRegression(C=10, max_iter=100000, solv...	
45	0.088587	0.003400	0.137018	0.002371	GaussianNB()	
44	0.090272	0.003864	0.144434	0.003029	GaussianNB()	
6	0.406218	0.044675	0.158792	0.004171	SVC(C=100, kernel='linear')	
18	0.097446	0.012525	0.127009	0.002279	LogisticRegression(C=100, max_iter=100000, pen...	
40	0.092801	0.003576	0.122844	0.004649	GaussianNB()	Trun
41	0.084573	0.001678	0.125302	0.006006	GaussianNB()	Trun
7	0.322189	0.033473	0.154420	0.002968	SVC(C=100, kernel='linear')	
0	0.988050	0.012398	0.135337	0.007808	SVC(C=100, kernel='linear')	Trun

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf	
31	0.085805	0.003062	0.122035	0.003597	LogisticRegression(C=10, max_iter=100000, solv...	
19	0.088043	0.003504	0.123575	0.006748	LogisticRegression(C=100, max_iter=100000, pen...	
12	0.093964	0.002072	0.120686	0.001711	LogisticRegression(C=100, max_iter=100000, pen...	Trun
1	1.054646	0.199691	0.140994	0.009476	SVC(C=100, kernel='linear')	Trun
13	0.090368	0.003660	0.117064	0.002256	LogisticRegression(C=100, max_iter=100000, pen...	Trun
30	0.120419	0.003628	0.197855	0.036804	LogisticRegression(C=10, max_iter=100000, solv...	
25	0.088963	0.002165	0.116476	0.004358	LogisticRegression(C=10, max_iter=100000, solv...	Trun
24	0.094061	0.013628	0.120814	0.009951	LogisticRegression(C=10, max_iter=100000, solv...	Trun
43	0.083787	0.001457	0.122642	0.005699	GaussianNB()	
42	0.088418	0.004502	0.124256	0.004905	GaussianNB()	
39	0.096562	0.016198	0.140491	0.033949	GaussianNB()	Trun
38	0.105749	0.016579	0.169837	0.036047	GaussianNB()	Trun
37	0.085616	0.001952	0.120355	0.006701	GaussianNB()	Trun
36	0.091900	0.012041	0.117752	0.004004	GaussianNB()	Trun

In [ ]:

```
from sklearn.pipeline import Pipeline
# Running the top 5 parameter combination pipelines on the test set and repo
'''
According to the 5-fold grid search cross validation, here are the best perf

1) Stemming, min_df = 5, NMF with 80 components, Logisitc Regression with L1

ion, min_df = 5, LSI with 80 components, Logistic Regression wi
```

Loading [MathJax]/extensions/Safe.js

```
3) Lemmatization, min_df = 3, NMF with 80 components, SVM with gamma_opt=100

4) Lemmatization, min_df = 3, LSI with 80 components, Logistic Regression wi

5) Lemmatization, min_df = 5, LSI with 80 components, SVM with gamma_opt=100
'''
gamma_opt = 100 # as computed earlier
L1_opt = 100 # as computed earlier
L2_opt = 10 # as computed earlier
pipeline1 = Pipeline([
    ('vect', CountVectorizer(min_df=5, stop_words='english')),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', NMF(n_components=80, init='random', random_state=42)),
    ('clf', LogisticRegression(C=L1_opt, penalty='l1', solver='liblinear', m
])

pipeline2 = Pipeline([
    ('vect', CountVectorizer(min_df=5, stop_words='english')),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', TruncatedSVD(n_components=80, n_iter=10, random_state=42)
    ('clf', LogisticRegression(C=L2_opt, penalty='l2', solver='liblinear', m
])

pipeline3 = Pipeline([
    ('vect', CountVectorizer(min_df=3, stop_words='english')),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', NMF(n_components=80, init='random', random_state=42)),
    ('clf', SVC(C=gamma_opt, kernel='linear')),
])

pipeline4 = Pipeline([
    ('vect', CountVectorizer(min_df=3, stop_words='english')),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', TruncatedSVD(n_components=80, n_iter=10, random_state=42)
    ('clf', LogisticRegression(C=L2_opt, penalty='l2', solver='liblinear', m
])

pipeline5 = Pipeline([
    ('vect', CountVectorizer(min_df=5, stop_words='english')),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', TruncatedSVD(n_components=80, n_iter=10, random_state=42)
    ('clf', SVC(C=gamma_opt, kernel='linear')),
])

# Train and evaluate accuracy on test set
pipeline1.fit(train_clean_stem, Y_train_label)
predict1 = pipeline1.predict(test_clean_stem)
print("Test Accuracy of 1st Best Parameter Combo:", metrics.accuracy_score(Y

pipeline2.fit(train_clean_lemm, Y_train_label)
predict2 = pipeline2.predict(test_clean_lemm)
print("Test Accuracy of 2nd Best Parameter Combo:", metrics.accuracy_score(Y

pipeline3.fit(train_clean_lemm, Y_train_label)
predict3 = pipeline3.predict(test_clean_lemm)
```

Loading [MathJax]/extensions/Safe.js

```
print("Test Accuracy of 3rd Best Parameter Combo:", metrics.accuracy_score(Y

pipeline4.fit(train_clean_lemm, Y_train_label)
predict4 = pipeline4.predict(test_clean_lemm)
print("Test Accuracy of 4th Best Parameter Combo:", metrics.accuracy_score(Y

pipeline5.fit(train_clean_lemm, Y_train_label)
predict5 = pipeline5.predict(test_clean_lemm)
print("Test Accuracy of 5th Best Parameter Combo:", metrics.accuracy_score(Y
```

```
Test Accuracy of 1st Best Parameter Combo: 0.9540229885057471
Test Accuracy of 2nd Best Parameter Combo: 0.9568965517241379
Test Accuracy of 3rd Best Parameter Combo: 0.9612068965517241
Test Accuracy of 4th Best Parameter Combo: 0.9612068965517241
Test Accuracy of 5th Best Parameter Combo: 0.9698275862068966
```

Q9 Multiclass Classification

- The confusion matrix, accuracy, recall, precision, and F1-score for the Naive Bayes Gaussian, OneVsOne SVM, and OneVsRest SVM classifiers are computed and reported below.
- To deal with class imbalanes, we were able to import SMOTE in order to oversample datapoints that were underrepresented in order to match the frequency of more popular labels.
- The confusion matrix is structured in such a way that has predicted classifications on the x-axis and true classifications on the y-axis. Therefore, if our classifier is doing a good job, then we should expect the diagonal going from the top-left-hand corner to the bottom-right-hand corner to be to be filled with greater counts compared to all other entries in the matrix. The confusion matrix allows us to easily detect where our model is running into confusions. For instance, if the distribution of predicted classifications for a given label is split between a given set of classes, we see that the model is having a hard time distinguishing between examples corresponding to that set of classes.
- All classifiers did a reasonably good job, and thus we observed distinct visible blocks on the major diagonal for the most part. The issue was with the 'heatwave' and 'forest fire' classes, which often got confused with each other.
- The 'heatwave' and 'forest fire' labels were merged into one larger label. We did this by labeling every instance of 'heatwave' as 'forest fire', which reduced the total number of labels from 10 to 9. Afterwards, the performance was evaluated again, and there was a significant boost in the accuracy of the model for both OneVsOne SVM and OneVsRest SVM (in both cases, a jump from roughly 80 percent to roughly 90 percent accuracy).
- This newly introduced imbalance did not have much of an impact on the effectiveness of the two multiclass SVM classifiers. To verify this, we used SMOTE to oversample underrepresented datapoints and then retrained the models using this modified version of the training data. For both types of SVM models, this had very little effect on the metrics we measured: accuracy, recall, precision, f1-score. These computations have

Loading [MathJax]/extensions/Safe.js d below.

```
In [ ]: print("Leaf labels:", list(set(df["leaf_label"])))
classes = ['basketball', 'baseball', 'tennis', 'football', 'soccer', 'forest

# Clean and process the multiclass data using Lemmatization and LSI
mc_train, mc_test = train_test_split(df[["full_text","leaf_label"]], test_si
mc_train_clean = mc_train["full_text"].apply(clean)
mc_test_clean = mc_test["full_text"].apply(clean)
mc_train_clean_lemm = mc_train_clean.apply(lemmatize_sentence)
mc_test_clean_lemm = mc_test_clean.apply(lemmatize_sentence)
vectorizer = CountVectorizer(min_df=3, stop_words='english')
mc_X_train = vectorizer.fit_transform(mc_train_clean_lemm)
mc_X_test = vectorizer.transform(mc_test_clean_lemm)
print(mc_X_train.shape)
print(mc_X_test.shape)
tfidf = TfidfTransformer()
mc_X_train_tfidf = tfidf.fit_transform(mc_X_train)
mc_X_test_tfidf = tfidf.transform(mc_X_test)
print(mc_X_train_tfidf.shape)
print(mc_X_test_tfidf.shape)
svd = TruncatedSVD(n_components=50, n_iter=10, random_state=42)
mc_X_train_svd = svd.fit_transform(mc_X_train_tfidf)
mc_X_test_svd = svd.transform(mc_X_test_tfidf)
print(mc_X_train_svd.shape)
print(mc_X_test_svd.shape)
```

```
Leaf labels: ['football', 'forest fire', 'baseball', 'basketball', 'socce
r', 'earthquake', 'drought', 'flood', 'heatwave', 'tennis']
(2780, 13254)
(696, 13254)
(2780, 13254)
(696, 13254)
```

```
In [ ]: # Convert categorical label to numerical labels (between 0 and 9)
mc_Y_train_label= mc_train["leaf_label"]
mc_Y_test_label = mc_test["leaf_label"]
for i in range(len(classes)):
  mc_Y_train_label[mc_Y_train_label  == classes[i]] = i
  mc_Y_test_label[mc_Y_test_label  == classes[i]] = i
mc_Y_train_label = mc_Y_train_label.astype(int)
mc_Y_test_label = mc_Y_test_label.astype(int)
```

```
In [ ]: # Multiclass Naive Bayes Gaussian Classifier

print("MULTICLASS NAIVE BAYES GAUSSIAN CLASSIFIER")
print("-" * 40)

mc_nb_classifier = GaussianNB()
mc_nb_prediction = mc_nb_classifier.fit(mc_X_train_svd, mc_Y_train_label).pr
confusion_matrix = metrics.confusion_matrix(mc_Y_test_label, mc_nb_predictic
print("Confusion Matrix:")
print(confusion_matrix)
print("-" * 40)
accuracy = metrics.accuracy_score(mc_Y_test_label, mc_nb_prediction)
print("Accuracy:", accuracy)
Loading [MathJax]/extensions/Safe.js tics.recall_score(mc_Y_test_label, mc_nb_prediction, average='ma
```

```
print("Recall:", recall)
precision = metrics.precision_score(mc_Y_test_label, mc_nb_prediction, avera
print("Precision:", precision)
f1 = metrics.f1_score(mc_Y_test_label, mc_nb_prediction, average='macro')
print("F1-Score:", f1)
```

```
MULTICLASS NAIVE BAYES GAUSSIAN CLASSIFIER
----------------------------------------
Confusion Matrix:
[[63  3  0  3  0  0  0  0  0  0]
 [ 0 56  1  2  2  0  0  0  0  1]
 [ 0 21 48  0  2  0  1  0  1  0]
 [ 2  5  0 62  3  0  0  0  0  0]
 [ 0  2  1  0 64  0  0  0  0  0]
 [ 0 12  1  0  1  4  1  0 22 30]
 [ 0  5  0  0  0  0 74  0  1  2]
 [ 0  9  4  0  0  0  0 60  0  0]
 [ 0  3  1  0  0  0  0  0 58  0]
 [ 0 12  3  0  0 14  4  0 15 17]]
----------------------------------------
Accuracy: 0.7270114942528736
Recall: 0.726785571628059
Precision: 0.7119712481034034
F1-Score: 0.7000930008035189
```

In [ ]:

```
# SVM Classifier One Vs One

print("MULTICLASS ONE VS ONE SVM CLASSIFIER")
print("-" * 40)

# Train the OneVsOneClassifier
from sklearn.multiclass import OneVsOneClassifier
mc_svm_oo = SVC(C=100, kernel='linear', probability=True)
mc_svm_oo = OneVsOneClassifier(mc_svm_oo)
mc_svm_oo_prediction = mc_svm_oo.fit(mc_X_train_svd, mc_Y_train_label).predi

# Calculate and print metrics
confusion_matrix = metrics.confusion_matrix(mc_Y_test_label, mc_svm_oo_predi
print("Confusion Matrix:")
print(confusion_matrix)
print("-" * 40)
accuracy = metrics.accuracy_score(mc_Y_test_label, mc_svm_oo_prediction)
print("Accuracy:", accuracy)
recall = metrics.recall_score(mc_Y_test_label, mc_svm_oo_prediction, average
print("Recall:", recall)
precision = metrics.precision_score(mc_Y_test_label, mc_svm_oo_prediction, a
print("Precision:", precision)
f1 = metrics.f1_score(mc_Y_test_label, mc_svm_oo_prediction, average='macro'
print("F1-Score:", f1)
```

```
MULTICLASS ONE VS ONE SVM CLASSIFIER
----------------------------------------
Confusion Matrix:
[[68  0  0  1  0  0  0  0  0  0]
 [ 0 57  1  2  1  1  0  0  0  0]
 [ 0  4 63  0  2  1  0  0  0  3]
 [ 1  3  0 67  1  0  0  0  0  0]
 [ 0  0  0  0 67  0  0  0  0  0]
 [ 0  2  1  0  1 14  1  1  1 50]
 [ 0  1  0  0  1  3 75  1  0  1]
 [ 0  4  0  0  0  2  0 67  0  0]
 [ 0  2  0  0  0  2  0  0 56  2]
 [ 0  2  4  0  0 32  2  2  1 22]]
----------------------------------------
Accuracy: 0.7988505747126436
Recall: 0.7969744148296428
Precision: 0.7940816212304116
F1-Score: 0.7937056226137214
```

In [ ]:

```python
# SVM Classifier for One Vs Rest

print("MULTICLASS ONE VS REST SVM CLASSIFIER")
print("-" * 40)

# Train the OneVsRestClassifier
from sklearn.multiclass import OneVsRestClassifier
mc_svm_or = SVC(C=100, kernel='linear', probability=True)
mc_svm_or = OneVsRestClassifier(mc_svm_or)
mc_svm_or_prediction = mc_svm_or.fit(mc_X_train_svd, mc_Y_train_label).predi

# Calculate and print metrics
confusion_matrix = metrics.confusion_matrix(mc_Y_test_label, mc_svm_or_predi
print("Confusion Matrix:")
print(confusion_matrix)
print("-" * 40)
accuracy = metrics.accuracy_score(mc_Y_test_label, mc_svm_or_prediction)
print("Accuracy:", accuracy)
recall = metrics.recall_score(mc_Y_test_label, mc_svm_or_prediction, average
print("Recall:", recall)
precision = metrics.precision_score(mc_Y_test_label, mc_svm_or_prediction, a
print("Precision:", precision)
f1 = metrics.f1_score(mc_Y_test_label, mc_svm_or_prediction, average='macro'
print("F1-Score:", f1)
```

```
MULTICLASS ONE VS REST SVM CLASSIFIER
----------------------------------------
Confusion Matrix:
[[68  0  0  1  0  0  0  0  0  0]
 [ 0 59  0  1  1  1  0  0  0  0]
 [ 0  7 59  0  2  1  1  0  1  2]
 [ 1  0  0 70  0  0  0  0  0  1]
 [ 0  0  0  0 67  0  0  0  0  0]
 [ 0  3  3  0  2  9  1  0  7 46]
 [ 0  1  1  0  1  2 74  1  2  0]
 [ 1  0  1  0  1  1  1 65  0  3]
 [ 0  1  2  0  0  0  0  0 59  0]
 [ 0  4  5  0  0 17  3  2  3 31]]
----------------------------------------
Accuracy: 0.8060344827586207
Recall: 0.806570807673055
Precision: 0.7830852135150221
F1-Score: 0.7893304995991937
```

In [ ]:

```
# Creating a subset
# Let's merge the following classes together into one label: 'heatwave' (co

mc_Y_train_merged_label = mc_Y_train_label
mc_Y_test_merged_label = mc_Y_test_label
mc_Y_train_merged_label[mc_Y_train_merged_label  == 9] = 5
mc_Y_test_merged_label[mc_Y_test_merged_label  == 9] = 5
```

In [ ]:

```
# SVM Classifier One Vs One Using New Subset

print("MULTICLASS ONE VS ONE SVM CLASSIFIER USING NEW SUBSET")
print("-" * 40)

# Train the OneVsOneClassifier
from sklearn.multiclass import OneVsOneClassifier
mc_svm_oo_subset = SVC(C=100, kernel='linear', probability=True)
mc_svm_oo_subset = OneVsOneClassifier(mc_svm_oo_subset)
mc_svm_oo_subset_prediction = mc_svm_oo_subset.fit(mc_X_train_svd, mc_Y_trai

# Calculate and print metrics
confusion_matrix = metrics.confusion_matrix(mc_Y_test_merged_label, mc_svm_o
print("Confusion Matrix:")
print(confusion_matrix)
print("-" * 40)
accuracy = metrics.accuracy_score(mc_Y_test_merged_label, mc_svm_oo_subset_p
print("Accuracy:", accuracy)
recall = metrics.recall_score(mc_Y_test_merged_label, mc_svm_oo_subset_predi
print("Recall:", recall)
precision = metrics.precision_score(mc_Y_test_merged_label, mc_svm_oo_subset
print("Precision:", precision)
f1 = metrics.f1_score(mc_Y_test_merged_label, mc_svm_oo_subset_prediction, a
print("F1-Score:", f1)
```

```
MULTICLASS ONE VS ONE SVM CLASSIFIER USING SUBSET
----------------------------------------
Confusion Matrix:
[[ 68   0   0   1   0   0   0   0   0]
 [  0  56   1   2   1   2   0   0   0]
 [  0   3  61   0   2   7   0   0   0]
 [  1   2   0  67   1   1   0   0   0]
 [  0   0   0   0  67   0   0   0   0]
 [  0   0   3   0   0 129   2   0   2]
 [  0   1   0   0   1   5  75   0   0]
 [  0   1   0   0   0   7   0  65   0]
 [  0   1   0   0   0   6   0   0  55]]
----------------------------------------
Accuracy: 0.9238505747126436
Recall: 0.9217307042159969
Precision: 0.9385846114688784
F1-Score: 0.9284984322218245
```

In [ ]:

```
# SVM Classifier One Vs Rest Using New Subset

print("MULTICLASS ONE VS REST SVM CLASSIFIER USING NEW SUBSET")
print("-" * 40)

# Train the OneVsRestClassifier
from sklearn.multiclass import OneVsRestClassifier
mc_svm_or_subset = SVC(C=100, kernel='linear', probability=True)
mc_svm_or_subset = OneVsRestClassifier(mc_svm_or_subset)
mc_svm_or_subset_prediction = mc_svm_or_subset.fit(mc_X_train_svd, mc_Y_trai

# Calculate and print metrics
confusion_matrix = metrics.confusion_matrix(mc_Y_test_merged_label, mc_svm_o
print("Confusion Matrix:")
print(confusion_matrix)
print("-" * 40)
accuracy = metrics.accuracy_score(mc_Y_test_merged_label, mc_svm_or_subset_p
print("Accuracy:", accuracy)
recall = metrics.recall_score(mc_Y_test_merged_label, mc_svm_or_subset_predi
print("Recall:", recall)
precision = metrics.precision_score(mc_Y_test_merged_label, mc_svm_or_subset
print("Precision:", precision)
f1 = metrics.f1_score(mc_Y_test_merged_label, mc_svm_or_subset_prediction, a
print("F1-Score:", f1)
```

```
MULTICLASS ONE VS REST SVM CLASSIFIER USING SUBSET
----------------------------------------
Confusion Matrix:
[[ 68   0   0   1   0   0   0   0   0]
 [  0  58   0   1   1   2   0   0   0]
 [  0   7  59   0   2   4   0   0   1]
 [  1   0   0  70   0   0   0   1   0]
 [  0   0   0   0  67   0   0   0   0]
 [  0   4   5   0   1 119   3   0   4]
 [  0   1   1   0   1   3  74   1   1]
 [  1   1   1   0   1   3   1  65   0]
 [  0   1   2   0   0   3   0   0  56]]
----------------------------------------
Accuracy: 0.9137931034482759
Recall: 0.9191675897105481
Precision: 0.9160904819557114
F1-Score: 0.916512459607979
```

In [ ]:

```
# Balancing the classes

from imblearn.over_sampling import SMOTE
oversample = SMOTE()
mc_X_train_svd_balanced, mc_Y_train_merged_label_balanced = oversample.fit_r

print(len(mc_X_train_svd))
print(len(mc_Y_train_merged_label))
print(len(mc_X_train_svd_balanced))
print(len(mc_Y_train_merged_label_balanced))
```

```
2780
2780
5076
5076
```

In [ ]:

```
print("MULTICLASS ONE VS ONE SVM CLASSIFIER USING NEW SUBSET - BALANCED")
print("-" * 40)

# Train the OneVsOneClassifier
from sklearn.multiclass import OneVsOneClassifier
mc_svm_oo_subset_balanced = SVC(C=100, kernel='linear', probability=True)
mc_svm_oo_subset_balanced = OneVsOneClassifier(mc_svm_oo_subset_balanced)
mc_svm_oo_subset_balanced_prediction = mc_svm_oo_subset_balanced.fit(mc_X_tr

# Calculate and print metrics
confusion_matrix = metrics.confusion_matrix(mc_Y_test_merged_label, mc_svm_c
print("Confusion Matrix:")
print(confusion_matrix)
print("-" * 40)
accuracy = metrics.accuracy_score(mc_Y_test_merged_label, mc_svm_oo_subset_b
print("Accuracy:", accuracy)
recall = metrics.recall_score(mc_Y_test_merged_label, mc_svm_oo_subset_balar
print("Recall:", recall)
precision = metrics.precision_score(mc_Y_test_merged_label, mc_svm_oo_subset
print("Precision:", precision)
f1 = metrics.f1_score(mc_Y_test_merged_label, mc_svm_oo_subset_balanced_prec
print("F1-Score:", f1)
```

```
MULTICLASS ONE VS ONE SVM CLASSIFIER USING SUBSET - BALANCED
----------------------------------------
Confusion Matrix:
[[ 68   0   0   1   0   0   0   0   0]
 [  0  56   1   2   1   2   0   0   0]
 [  0   3  62   3   2   3   0   0   0]
 [  1   1   0  68   1   1   0   0   0]
 [  0   0   0   0  67   0   0   0   0]
 [  0   4   5   0   1 120   3   1   2]
 [  0   2   0   0   1   5  74   0   0]
 [  0   2   0   0   1   3   0  67   0]
 [  0   2   0   0   0   3   0   1  56]]
----------------------------------------
Accuracy: 0.9166666666666666
Recall: 0.9209242841069365
Precision: 0.9216754867238343
F1-Score: 0.9202466636185971
```

In [ ]:

```
print("MULTICLASS ONE VS REST SVM CLASSIFIER USING NEW SUBSET - BALANCED")
print("-" * 40)

# Train the OneVsRestClassifier
from sklearn.multiclass import OneVsRestClassifier
mc_svm_or_subset_balanced = SVC(C=100, kernel='linear', probability=True)
mc_svm_or_subset_balanced = OneVsRestClassifier(mc_svm_or_subset_balanced)
mc_svm_or_subset_balanced_prediction = mc_svm_or_subset_balanced.fit(mc_X_tr

# Calculate and print metrics
confusion_matrix = metrics.confusion_matrix(mc_Y_test_merged_label, mc_svm_c
print("Confusion Matrix:")
print(confusion_matrix)
print("-" * 40)
accuracy = metrics.accuracy_score(mc_Y_test_merged_label, mc_svm_or_subset_b
print("Accuracy:", accuracy)
recall = metrics.recall_score(mc_Y_test_merged_label, mc_svm_or_subset_balar
print("Recall:", recall)
precision = metrics.precision_score(mc_Y_test_merged_label, mc_svm_or_subset
print("Precision:", precision)
f1 = metrics.f1_score(mc_Y_test_merged_label, mc_svm_or_subset_balanced_prec
print("F1-Score:", f1)
```

```
MULTICLASS ONE VS REST SVM CLASSIFIER USING SUBSET - BALANCED
----------------------------------------
Confusion Matrix:
[[ 68   0   0   1   0   0   0   0   0]
 [  0  57   2   1   1   0   1   0   0]
 [  0   3  63   1   3   3   0   0   0]
 [  1   0   0  70   0   0   0   1   0]
 [  0   0   0   0  67   0   0   0   0]
 [  0   6   8   0   1 112   3   1   5]
 [  0   1   1   0   1   3  74   1   1]
 [  1   0   2   0   1   1   1  66   1]
 [  0   1   2   0   0   1   0   0  58]]
----------------------------------------
Accuracy: 0.9123563218390804
Recall: 0.922851100233626
Precision: 0.9111708126404364
F1-Score: 0.9158741367174122
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
import pickle
from collections import Counter, defaultdict
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag, word_tokenize

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score # recall scorer
from sklearn.metrics import precision_score # precision scorer
from sklearn.metrics import roc_curve # ROC curve
from sklearn.metrics import f1_score # f1

from sklearn import *
from sklearn.svm import SVC, LinearSVC
import itertools

import os
import gensim
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import KeyedVectors
from scipy import spatial

from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC, LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.decomposition import TruncatedSVD, NMF

import umap
import umap.plot

# used to cache results
from tempfile import mkdtemp
from shutil import rmtree
import joblib
from joblib import Memory
```

Question 10 (1) GLoVE embeddings allows for a more nuanced and effective capture of word relationships, better handling of word frequency disparities, and more efficient and robust model training. Direct co-occurrence probabilities are heavily influenced by the frequency of words. Some common words occured more often with many other words, which are overshadow more meaningful co-occurrences. And the ratio of co-occurrence probabilities are heavily dependent on the frequency of words. Ratios help in differentiating between words that are used in similar contexts versus those that are genuinely related.

(2)No, we have different vector for "running" in this case. The neighbor words will play a very important role in the vector representation of the word.

In [3]:
```
embeddings_dict = {}
dimension_of_glove = 300
with open("glove/glove.6B.300d.txt", 'r', encoding='utf-8') as f:

    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], "float32")
        embeddings_dict[word] = vector
```

In [4]:
```
print(np.linalg.norm(embeddings_dict['woman']-embeddings_dict['man']))
print(np.linalg.norm(embeddings_dict['wife']-embeddings_dict['husband']))
print(np.linalg.norm(embeddings_dict['wife']-embeddings_dict['orange']))
```

```
4.7539396
3.1520464
8.667715
```

(3) ||GLoVE["woman"] - GLoVE["man"]||2 = 4.7539396 ||GLoVE["wife"] - GLoVE["husband"]||2 = 3.1520464 ||GLoVE["wife"] - GLoVE["orange"] = 8.667715 The value for the first line is very similar to second line. The "husband" is not similar with word "orange" because they have 5.51 similarity score. we also observe that the closest word for "woman" is "woman", "girl" ,and "man" (4) The choice between stemming and lemmatization depends on the specific requirements of the task and the nature of the text data.Stemming is a more rudimentary process that chops off the ends of words in the hope of achieving the goal correctly most of the time. It's faster and more straightforward. Useful in tasks where the complexity of the language is less important. Lemmatization involves a more sophisticated analysis of the word to return it to its dictionary form. It takes into consideration the word's part of speech, its meaning in context, and its grammatical usage. Therefor,Lemmatization is chosen over stemming.

In [5]:
```
root_folder='.'
glove_folder_name='glove'
glove_filename='glove.6B.300d.txt'
glove_path = os.path.abspath(os.path.join(root_folder, glove_folder_name, gl
word2vec_output_file = glove_filename+'.word2vec'
glove2word2vec(glove_path, word2vec_output_file)
model = KeyedVectors.load_word2vec_format(word2vec_output_file, binary=False
```

```
C:\Users\josep\AppData\Local\Temp\ipykernel_14628\1082466826.py:6: Deprecat
ionWarning: Call to deprecated `glove2word2vec` (KeyedVectors.load_word2vec
_format(.., binary=False, no_header=True) loads GLoVE text vectors.).
  glove2word2vec(glove_path, word2vec_output_file)
```

In [6]:
```
def find_closest_embeddings(embedding):
    return sorted(embeddings_dict.keys(), key=lambda word: spatial.distance.

print(find_closest_embeddings(embeddings_dict["woman"])[:3])
```

Loading [MathJax]/extensions/Safe.js

```
print(find_closest_embeddings(embeddings_dict["man"])[:3])
print(find_closest_embeddings(embeddings_dict["wife"])[:3])
print(find_closest_embeddings(embeddings_dict["man"])[:3])
```

```
['woman', 'girl', 'man']
['man', 'woman', 'person']
['wife', 'husband', 'daughter']
['man', 'woman', 'person']
```

Question 11

In [7]:
```
df = pd.read_csv("Project1-ClassificationDataset.csv")
print('Number of data points : ', df.shape[0])
print('Number of features : ', df.shape[1])
```

```
Number of data points :  3476
Number of features :  8
```

In [8]:
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df[["full_text","root_label"]], test_size=0.2
print('Number of points in train data:', train.shape[0])
print('Number of points in test data:', test.shape[0])
```

```
Number of points in train data: 2780
Number of points in test data: 696
```

In [9]:
```
import re
def clean(text):
    text = re.sub(r"http\S+", '', 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)
    texter = re.sub('\r', ' ',texter)
    texter = re.sub(r"[0-9]","", texter)
    texter = re.sub('[^a-zA-Z0-9\n]', ' ', texter)
    texter = re.sub('\s+',' ', texter)
    texter = texter.lower()
    clean = re.compile('<.*?>')
    texter = texter.encode('ascii', 'ignore').decode('ascii')
    texter = re.sub(clean, '', texter)
    if texter == "":
        texter = ""
    return texter
```

In [10]:
```
X_train = train['full_text'].apply(clean)
X_test = test['full_text'].apply(clean)
X_train.head()
```

Out[10]:
```
16      we re down to the final four after more than ...
2451    sacramento county another earthquake has occu...
1840    this article has been reviewed according to s...
2250    leominster right now david ed good evening th...
2657    at least kids across states have gotten sick ...
Name: full_text, dtype: object
```

In [11]:
```
X_test.head()
```

Out[11]:
```
2125    close get email notifications on subject dail...
2058    this story is part of nature s an annual list...
13      oklahoma city began the nba preseason with a ...
2400    public works crews in pacifica were called in...
10      season tips off with access to out of market ...
Name: full_text, dtype: object
```

In [12]:
```
y_train_encoded = train["root_label"].copy()
y_test_encoded = test["root_label"].copy()

y_train_encoded[y_train_encoded == 'sports'] = 0
y_test_encoded[y_test_encoded == 'sports'] = 0

y_train_encoded[y_train_encoded== 'climate'] = 1
y_test_encoded[y_test_encoded == 'climate'] = 1

print("Training Set\n")
print("Original train_dataset:\n" + str(train["root_label"][0:20]))
print("\nBinarized train_dataset:\n" + str(y_train_encoded[0:20]))
print("\nTest Set\n")
print("Original test_dataset:\n" + str(test["root_label"][0:20]))
print("\nBinarized test_dataset:\n" + str(y_test_encoded[0:20]))
```

```
Training Set

Original train_dataset:
16       sports
2451    climate
1840    climate
2250    climate
2657    climate
2199    climate
616      sports
1101     sports
2853    climate
2553    climate
3193    climate
1317     sports
664      sports
1264     sports
2898    climate
970      sports
161      sports
880      sports
3459    climate
942      sports
Name: root_label, dtype: object

Binarized train_dataset:
16      0
2451    1
1840    1
2250    1
2657    1
2199    1
616     0
1101    0
2853    1
2553    1
3193    1
1317    0
664     0
1264    0
2898    1
970     0
161     0
880     0
3459    1
942     0
Name: root_label, dtype: object

Test Set

Original test_dataset:
2125    climate
2058    climate
13       sports
2400    climate
            ts
```

```
747      sports
816      sports
1919    climate
3352    climate
3146    climate
441      sports
278      sports
3384    climate
2984    climate
804      sports
2143    climate
2499    climate
3007    climate
2024    climate
2719    climate
Name: root_label, dtype: object

Binarized test_dataset:
2125    1
2058    1
13      0
2400    1
10      0
747     0
816     0
1919    1
3352    1
3146    1
441     0
278     0
3384    1
2984    1
804     0
2143    1
2499    1
3007    1
2024    1
2719    1
Name: root_label, dtype: object
```

In [13]:

```
class Word2VecVectorizer:
    def __init__(self, model):
        print("Loading in word vectors...")
        self.word_vectors = model
        print("Finished loading in word vectors")

    def fit(self, data):
        pass

    def transform(self, data):
        v = self.word_vectors.get_vector('king')
        self.D = v.shape[0]

        X = np.zeros((len(data), self.D))
        n = 0
        [illegible]count = 0
```

Loading [MathJax]/extensions/Safe.js

```python
        for sentence in data:
            tokens = sentence.split()
            vecs = []
            m = 0
            for word in tokens:
                try:
                    vec = self.word_vectors.get_vector(word)
                    vecs.append(vec)
                    m += 1
                except KeyError:
                    pass
            if len(vecs) > 0:
                vecs = np.array(vecs)
                X[n] = vecs.mean(axis=0)
            else:
                emptycount += 1
            n += 1
        print("Number of samples with no words found: %s / %s" % (emptycount
        return X

    def fit_transform(self, data):
        self.fit(data)
        return self.transform(data)
```

In [14]:
```python
vectorizer = Word2VecVectorizer(model)
```

```
Loading in word vectors...
Finished loading in word vectors
```

In [15]:
```python
X_train_fit = vectorizer.fit_transform(X_train)
y_train = y_train_encoded.astype(str).astype(int)
X_test_fit = vectorizer.transform(X_test)
y_test = y_test_encoded.astype(str).astype(int)
print(X_train_fit.shape,X_test_fit.shape)
```

```
Number of samples with no words found: 0 / 2780
Number of samples with no words found: 0 / 696
(2780, 300) (696, 300)
```

In [16]:
```python
def train_svm_with_gridsearch(X_train, y_train, X_test):
    clf_cv = svm.SVC(random_state=42)
    param_grid = {
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 200, 400, 600, 800, 1000],
        'kernel': ['linear']
    }

    grid_search = GridSearchCV(clf_cv, param_grid, cv=5, scoring='accuracy',
    grid_search.fit(X_train, y_train)
    y_pred = grid_search.best_estimator_.predict(X_test)

    return y_pred, grid_search

# Use the function and get the grid_search object
y_pred_glove, grid_search = train_svm_with_gridsearch(X_train_fit, y_train,
print(grid_search.best_estimator_)
```

Loading [MathJax]/extensions/Safe.js

```
nel='linear', random_state=42)
```

In [17]:
```
print("Accuracy (Best GLoVE classifier):", accuracy_score(y_test,y_pred_glov
print("Recall (Best GLoVE classifier):", recall_score(y_test,y_pred_glove))
print("Precision (Best GLoVE classifier):", precision_score(y_test,y_pred_gl
print("F1-Score (Best GLoVE classifier):", f1_score(y_test,y_pred_glove))
```

```
Accuracy (Best GLoVE classifier): 0.9583333333333334
Recall (Best GLoVE classifier): 0.961218836565097
Precision (Best GLoVE classifier): 0.9585635359116023
F1-Score (Best GLoVE classifier): 0.9598893499308437
```

(1)Feature Engineering Process Using GLoVE Embeddings Text Preprocessing: cleaning: remove punctuation, special characters, and possibly stop words. Lemmatization:choose to lemmatize the words to ensure they match the form most likely found in the GLoVE embeddings.We convert each sentence into vectors. All the features which are described in the vectors and that are significant are based in the pre-trained GLoVE embeddingsfile.As a final step, we normalize the final vectors (2)Linear SVM classifier model is chosen (10) to train and evaluate it with the GLoVE-based feature Accuracy (Best GLoVE classifier): 0.9583333333333334 Recall (Best GLoVE classifier): 0.961218836565097 Precision (Best GLoVE classifier): 0.9585635359116023 F1-Score (Best GLoVE classifier): 0.9598893499308437

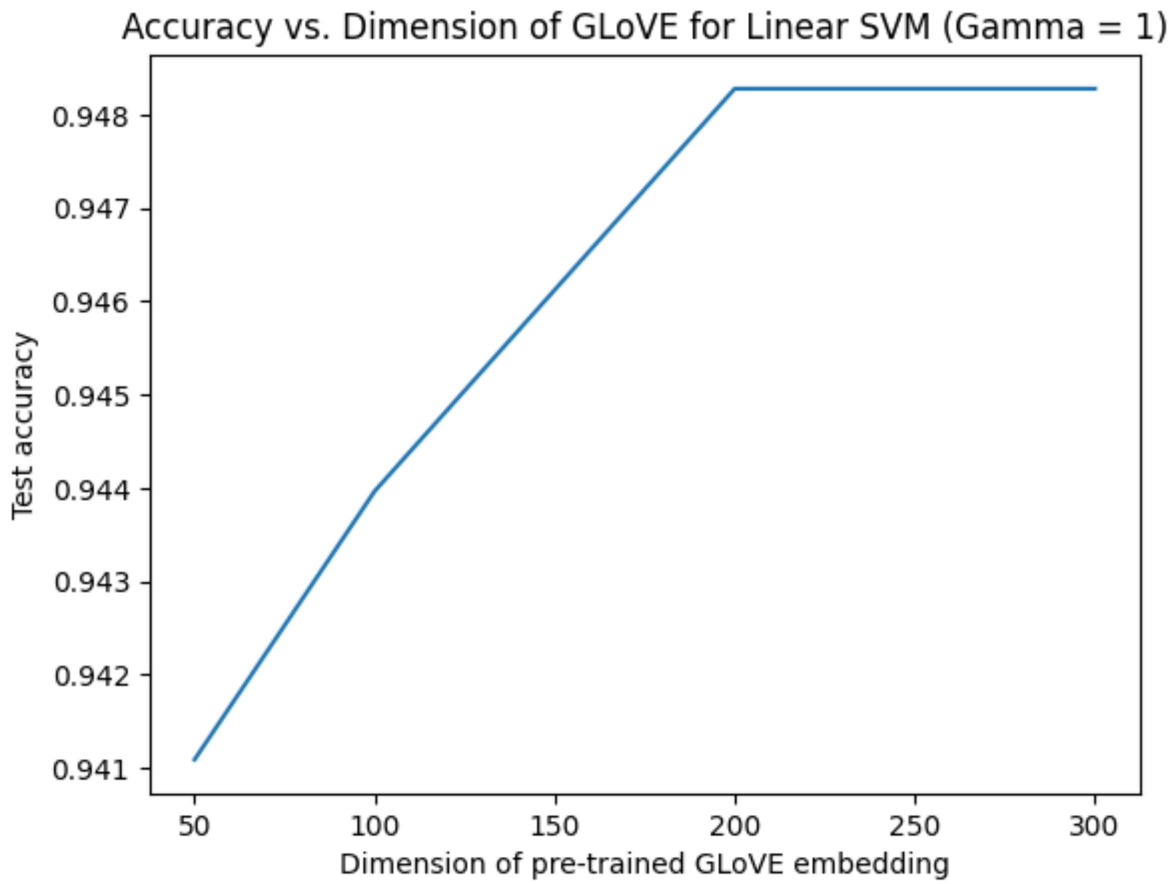
In [18]:
```
def load_glove_model(glove_file_path):
    # Load GloVe model directly if possible
    return KeyedVectors.load_word2vec_format(glove_file_path, binary=False,

def train_and_evaluate_svm(X_train, y_train, X_test, y_test):
    clf = svm.SVC(kernel='linear', C=1, random_state=42)
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
    return accuracy_score(y_test, predictions)

root_folder = '.'
glove_folder_name = 'glove'
filenames_glove = ['glove.6B.50d.txt', 'glove.6B.100d.txt', 'glove.6B.200d.t
accu_list_glove = []

y_train = y_train_encoded.astype(str).astype(int)
y_test = y_test_encoded.astype(str).astype(int)

for filename in filenames_glove:
    print('Training for:', filename)
    glove_path = os.path.abspath(os.path.join(root_folder, glove_folder_name

    model = load_glove_model(glove_path)
    vectorizer = Word2VecVectorizer(model)
    X_train_fit = vectorizer.fit_transform(X_train)
    X_test_fit = vectorizer.transform(X_test)

    accuracy = train_and_evaluate_svm(X_train_fit, y_train, X_test_fit, y_te
    accu_list_glove.append(accuracy)

print("Accuracies:", accu_list_glove)
```

```
Training for: glove.6B.50d.txt
Loading in word vectors...
Finished loading in word vectors
Number of samples with no words found: 0 / 2780
Number of samples with no words found: 0 / 696
Training for: glove.6B.100d.txt
Loading in word vectors...
Finished loading in word vectors
Number of samples with no words found: 0 / 2780
Number of samples with no words found: 0 / 696
Training for: glove.6B.200d.txt
Loading in word vectors...
Finished loading in word vectors
Number of samples with no words found: 0 / 2780
Number of samples with no words found: 0 / 696
Training for: glove.6B.300d.txt
Loading in word vectors...
Finished loading in word vectors
Number of samples with no words found: 0 / 2780
Number of samples with no words found: 0 / 696
Accuracies: [0.9410919540229885, 0.9439655172413793, 0.9482758620689655, 0.
9482758620689655]
```

In [19]:

```
print("Accuracies:", accu_list_glove)
dim_list = [50,100,200,300]
plt.plot(dim_list,accu_list_glove)
plt.title('Accuracy vs. Dimension of GLoVE for Linear SVM (Gamma = 1)')
plt.xlabel('Dimension of pre-trained GLoVE embedding')
plt.ylabel('Test accuracy')
plt.show()
```

```
Accuracies: [0.9410919540229885, 0.9439655172413793, 0.9482758620689655, 0.
9482758620689655]
```



(12)Accuracy vs Dimension plot for GLoVE using Linear SVM as a classifier.We can tell that when the dimension of the GLoVE embedding increases, the accuracy of the test set will also incease. Higer dimension providing us with better feature dependencies and a more accurate model

In [20]:
```
reduced_dim_embedding = umap.UMAP(n_components=2, metric='euclidean').fit(X_
print(reduced_dim_embedding.embedding_.shape)
```

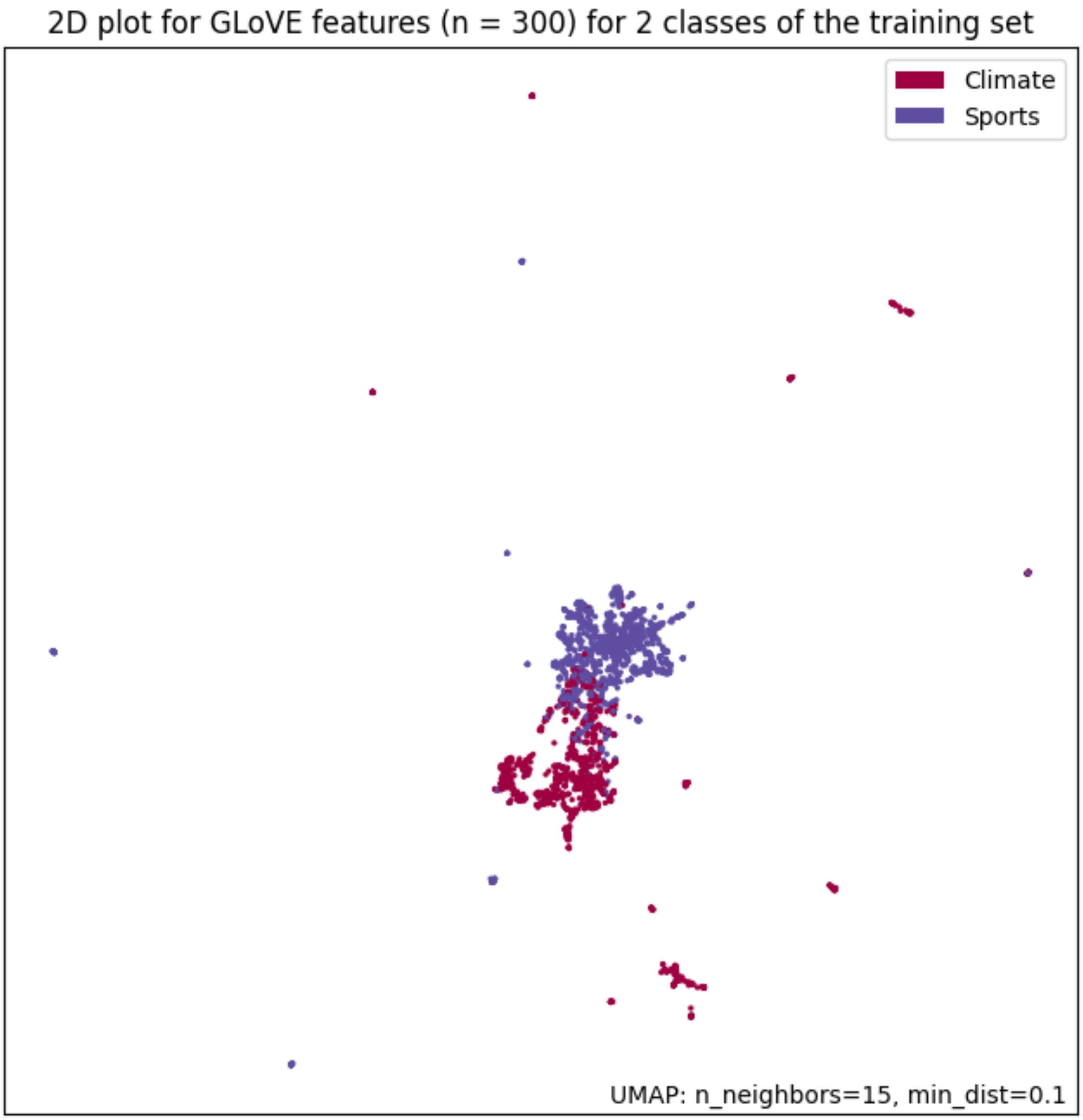
(2780, 2)

In [21]:
```
# Fit UMAP for your training data
umap_model = umap.UMAP(n_components=2, metric='euclidean')
umap_model.fit(X_train_fit)

# Prepare labels for the training set
YtrainTextLabel = []
for label in y_train:
    if label == 0:
        YtrainTextLabel.append('Sports')
    else:
        YtrainTextLabel.append('Climate')

# Plotting for GloVe features
f = umap.plot.points(umap_model, labels=np.array(YtrainTextLabel))
plt.title('2D plot for GLoVE features (n = 300) for 2 classes of the trainin

# Fit UMAP for normalized random vectors
```

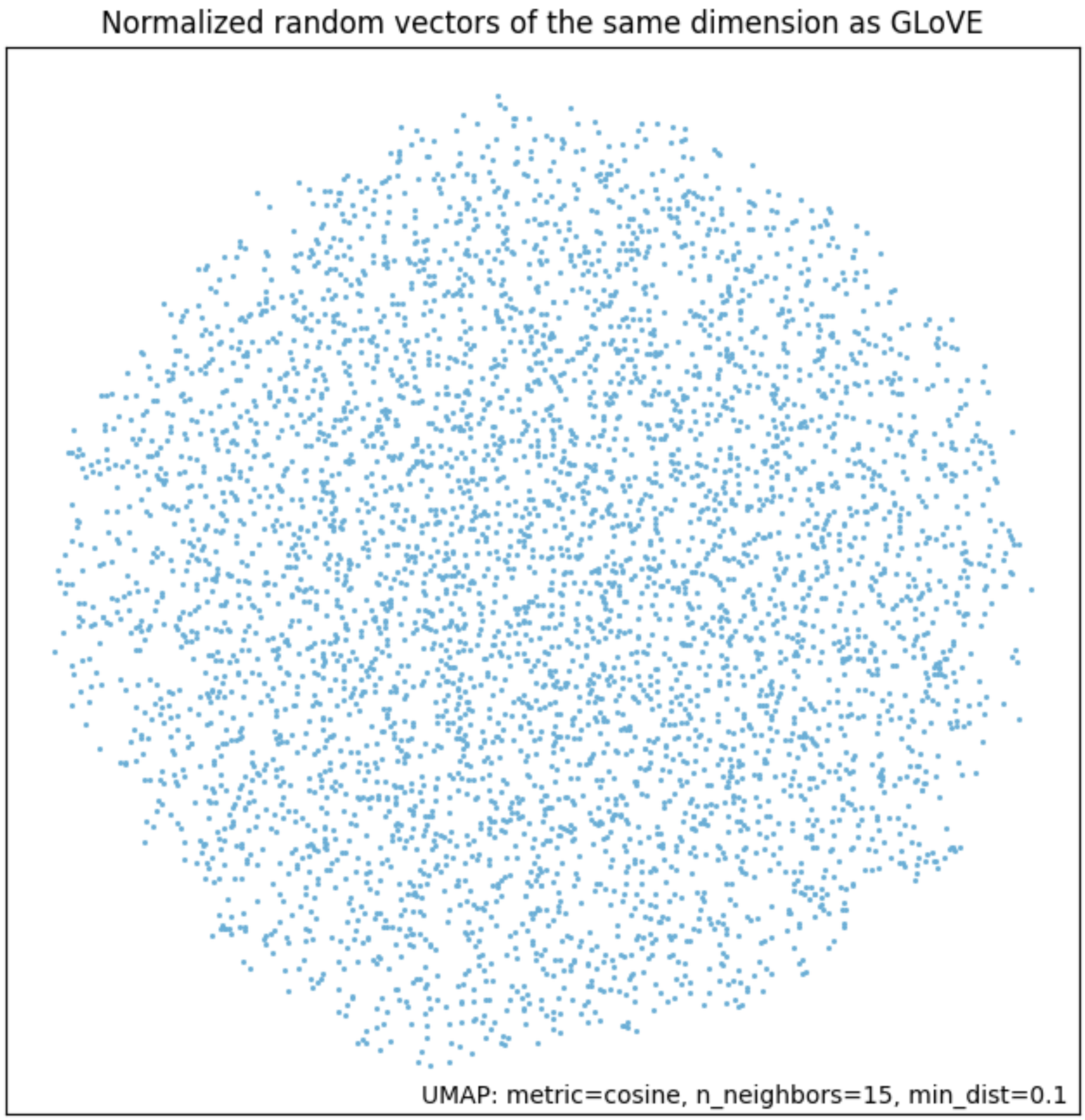
Loading [MathJax]/extensions/Safe.js

```python
s = np.random.normal(0, 1, [4800, 300])
s = s / np.linalg.norm(s, axis=1, keepdims=True)
umap_model_s = umap.UMAP(n_components=2, metric='cosine')
umap_model_s.fit(s)

# Plotting for normalized random vectors
g = umap.plot.points(umap_model_s)
plt.title('2D plot normalized random vectors')



######################

plt.title("Normalized random vectors of the same dimension as GLoVE")
plt.show()
```

```
C:\Users\josep\AppData\Local\Programs\Python\Python311\Lib\site-packages\um
ap\plot.py:449: UserWarning: *c* argument looks like a single numeric RGB o
r RGBA sequence, which should be avoided as value-mapping will have precede
nce in case its length matches with *x* & *y*.  Please use the *color* keyw
ord-argument or provide a 2D array with a single row if you intend to speci
fy the same RGB or RGBA value for all points.
  ax.scatter(points[:, 0], points[:, 1], s=point_size, c=color)
```



Loading [MathJax]/extensions/Safe.js



Question 13 On visualizing the 2D plots for GLoVE and random vectors with the same dimension as the GLoVE embeddings, distinct clusters are formed only in the GLoVE model and not for the random vectors.