Project 1: End-to-End Pipeline to Classify News Articles

Large-Scale Data Mining: Models and Algorithms

Ananya Deepak Deoghare, Hariram Veeramani, Madhav Sankar Krishnakumar

005627628, 005528336, 405692669

ECE 219 Winter 2022

Question 1

How many rows (samples) and columns (features) are present in the dataset? We got about 2000 samples.

We were provided a customized dataset. The dataset consisted of various newspaper articles, along with other details such as keywords, published date, leaf-labels, etc. The column details are provided below. Number of rows: 2072 and number of columns: 9

```
Range Index [Rows]: 2072 entries, 0 to 2071
Data columns (total 9 columns)
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Unnamed: 0     2072 non-null   int64
 1   full_text      2072 non-null   object
 2   summary        2072 non-null   object
 3   keywords       2072 non-null   object
 4   publish_date   1125 non-null   object
 5   authors        2072 non-null   object
 6   url            2072 non-null   object
 7   leaf_label     2072 non-null   object
 8   root_label     2072 non-null   object
```

Histograms: Plot 3 histograms on : (a) The total number of alpha-numeric characters per data point (row) in the feature full text: i.e count on the x-axis and frequency on the y-axis; (b) The column leaf label – class on the x-axis; (c) The column root label – class on the x-axis. Interpret Plots: Provide qualitative interpretations of the histograms.

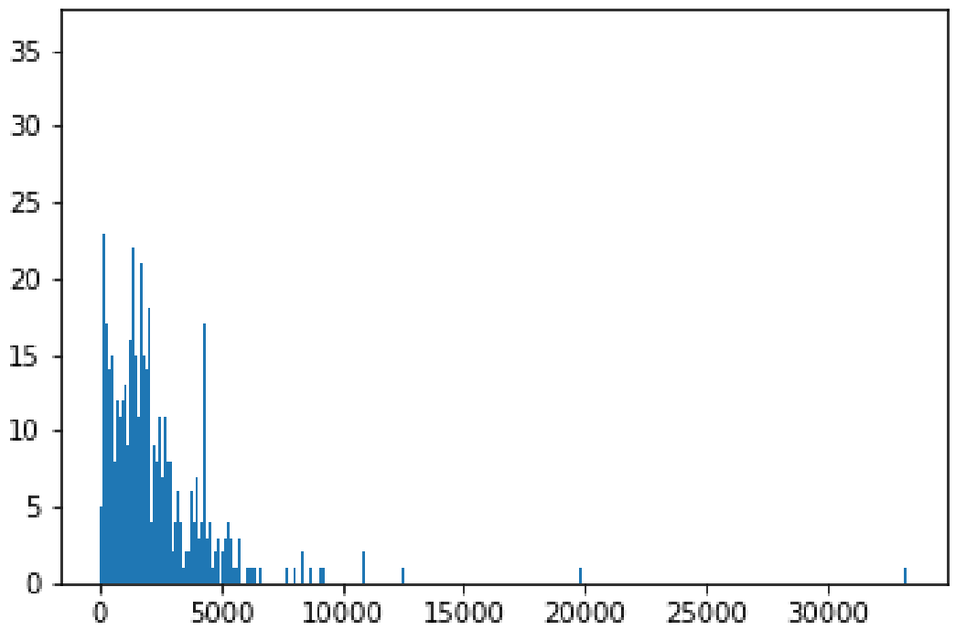

Figure 1: The total number of alpha-numeric characters per data point (row) in the feature full text: i.e count on the x-axis and frequency on the y-axis

In Fig 1., we can notice that the maximum frequency of alphanumeric character was 26. The number of alphanumeric characters vary across the documents. The alphanumeric characters vary from 40 to 30,000 characters. This shows that the data set is very diverse in terms of the size. Therefore we need to normalize the data before using it.

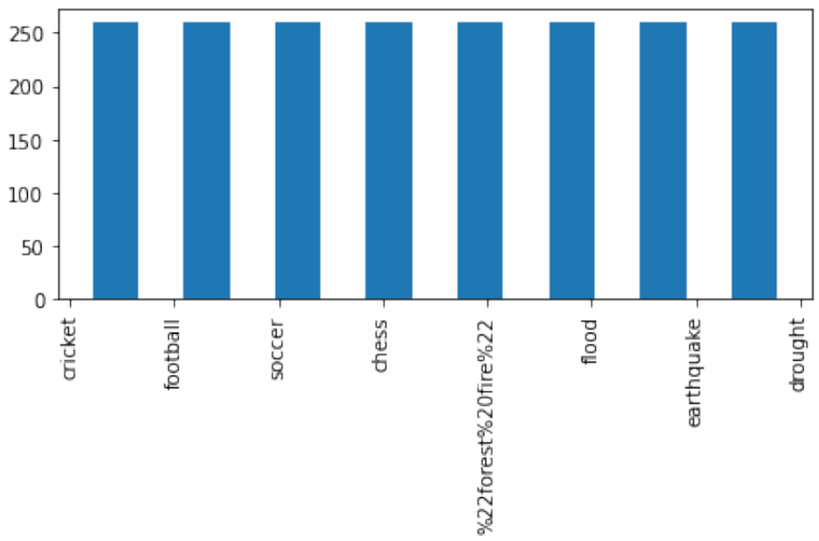

Figure 2: The column leaf label class (categories) on the x-axis and number of documents on the y-axis

Figure 3: The column root label class (categories) on the x-axis and number of documents on the y-axis

Fig. 2 and Fig. 3 are used to show the class wise distribution of the data. We notice that the data is equally distributed between all the root labels and leaf labels. This proves that there is no *class imbalance*, and we can directly apply the classification models. If class imbalance was present, we would have to either oversample or undersample the data.

Question 2

Report the number of training and testing samples.
In order to measure the performance of the binary classifier model, we split the dataset into a training and a testing set. The model is trained on the training set and evaluated on the testing set. We set random seed as **42** and the fraction of the test set as **0.2**.
Number of Training Samples: **1657**
Number of Testing Samples: **415**

Question 3

Use the following specs to extract features from the textual data:
• Before doing anything, please clean each data sample using the code block provided above. This function helps remove many but not all HTML artefacts from the crawler's output. You can also build your own cleaning module if you find this function to be ineffective.
• Use the "english" stopwords of the CountVectorizer 4
• Exclude terms that are numbers (e.g. "123", "-45", "6.7" etc.)
• Perform lemmatization with nltk.wordnet.WordNetLemmatizer and pos tag • Use min df=3
We used the below specifications provided to us to clean the data.

- Used the Code Snippet given to clean the data.

- Excluded terms that are numbers (e.g. “123”, “-45”, “6.7” etc.). Regex matching and char.isdigit() are used to remove these numbers.
- We Perform lemmatization with nltk.wordnet.WordNetLemmatizer that made use of the POS (part of speech) tag. We converted the given text into sentences and applied lemmatization one sentence at a time. This ensures contextual POS tagging and hence more accurate lemmatization. Punctuations were removed after lemmatization as the text had to be split into sentences. Lemmatization helps to bring down the number of features as all the words with the same root lemma are grouped together. This helps to improve the model training and testing time.
- We used CountVectorizer() to convert the lemmatized text into a bag-of-words matrix. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.
  To omit common words in the English language (called stop-words) that do not provide any contextual information such as “and”, “the”, “I” etc., we set the argument stopwords to “english”.
- We used min df=3. This ensures that words that appear less than 3 times in the training data are dropped. Therefore rare words will be dropped and this helps to reduce the feature set size.
- Then finally the TfidfTransformer() is applied to convert it into Term Frequency Inverse Document Frequency matrix. As the name explains, it helps to find words within the text that helps to differentiate it from the other texts. That is there could be some words that appear uniformly across all documents and some other ones that appear only in select types of documents. The latter words are more important to us in a classification problem.

```
import re
import string
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk import pos_tag, word_tokenize, sent_tokenize
from nltk.corpus import wordnet
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('punkt')

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

def remove_numbers(text):
    text_nonum = re.sub(r'\b[0-9]+\b\s*', '', text)
    clean_text = []
    for word in word_tokenize(text):
        if (not any(char.isdigit() for char in word)):
            clean_text.append(word)
    return (" ").join(clean_text)

def penn2morphy(penntag):
    morphy_tag = {'NN':'n', 'JJ':'a',
                  'VB':'v', 'RB':'r'}
    try:
        return morphy_tag[penntag[:2]]
    except:
        return 'n'

def lemmatized(text):
    lemmatizer = WordNetLemmatizer()

    sent_pos_tags = [pos_tag(word_tokenize(sent)) for sent in sent_tokenize(text)]
    pos_tags = []
    for sentence_tags in sent_pos_tags:
        for tags in sentence_tags:
            pos_tags.append(tags)

    lemmatized_words = []
    lemm_text = [lemmatizer.lemmatize(word, pos = penn2morphy(tag)) for word, tag in pos
    for lemma in lemm_text:
        if (not any(char in lemma for char in string.punctuation)):
            lemmatized_words.append(lemma.lower())
    return lemmatized_words

def stem(text):
  stemmer = PorterStemmer()
  stem_text = [stemmer.stem(word) for word in word_tokenize(text)]
  return " ".join(stem_text)

def preprocess(text):
  text = clean(text)
  text = remove_numbers(text)
```

```
  text = lemmatized(text)
  return ' '.join(text)

print(train['full_text'])
train['clean'] = train['full_text'].apply(lambda x:preprocess(x))
test['clean'] = test['full_text'].apply(lambda x:preprocess(x))
print(train['clean'])
```

What are the pros and cons of lemmatization versus stemming? How do these processes affect the dictionary size?

Both lemmatization and stemming are word normalizing techniques. 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. Lemmatization is a more sophisticated version of stemming. It reduces each word to its proper base form, that is, a word that we can find in a dictionary.

Dictionary size will be smaller for lemmatization when compared with stemming as the words will be mapped correctly to the roots. For instance, take words studying and studies. Lemmatization converts both to study while stemming converts them study and studi respectively, thereby increasing the dictionary size.

min df means minimum document frequency. How does varying min df change the TF-IDF matrix?

min_df is used for removing terms that appear too infrequently i.e., the rare words. min_df = 3 means "ignore terms that appear in less than 3 documents". Decreasing the value of min_df increases the size of the resulting bag-of-words matrix.

Should I remove stopwords before or after lemmatizing? Should I remove punctuations before or after lemmatizing? Should I remove numbers before or after lemmatizing?

We should remove stopwords before lemmatizing to reduce the computational cost.

We removed the punctuations after lemmatizing as the text was converted to sentences before POS tagging. This helps improve the POS tag accuracy and thereby provides better lemmatization results.

We removed the numbers before lemmatizing to reduce the computational cost.

Report the shape of the TF-IDF-processed train and test matrices. The number of rows should match the results of Question 2. The number of columns should roughly be in the order of $k * 10^3$. This dimension will vary depending on your exact method of cleaning and lemmatizing and that is okay.

The shape of the TF-IDF-processed train and test matrices are given below. The number of rows match the results from Question 2.

```
TF-IDF Processed train matrix shape: (1657, 10054)
TF-IDF Processed test matrix shape: (415, 10054)
```

Question 4

Plot the explained variance ratio across multiple different k = [1, 10, 50, 100, 200, 500, 1000, 2000] for LSI and for the next few sections choose k = 50. What does the explained variance ratio plot look like? What does the plot's concavity suggest?

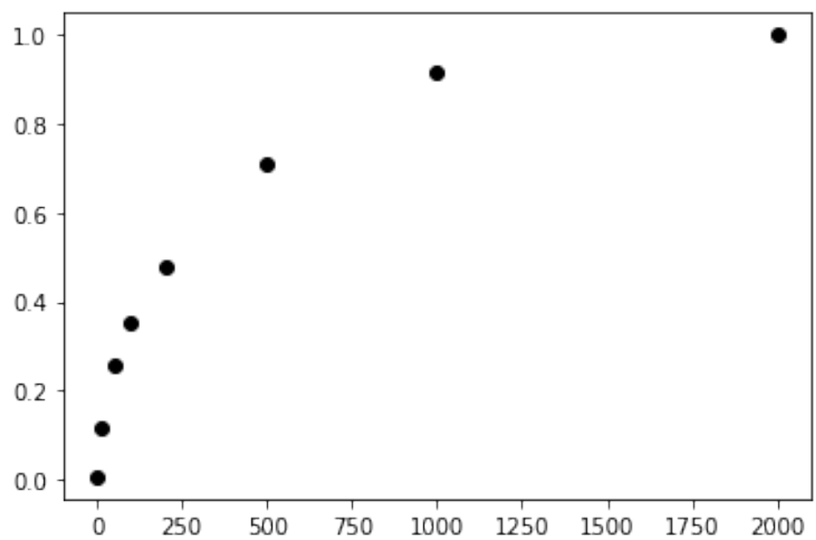

Figure 4: Explained variance ratio across multiple k's for LSI

The x axis of the Fig 4. represents the values of k, and the y axis has the variance ratio value. The explained variance ratio is the percentage of variance that is attributed by each of the selected components.
For a given k, LSI chooses the top k components in the descending order of the variance ratio that the component explains. We notice that around 80% of the variance is covered by around 1000 features i.e. k = 1000. The plot's concavity suggests that as we increase the values of k, the variance also increases up to a point (100%) but the increase gets slower with larger k. For example, the difference between the variance explained for k=50 and k=100 is more than that of k=1950 and k=2000. This makes sense as the components are chosen in decreasing order of their variance explanation ratio and the fact that it is not possible to go beyond 100% in the variance explanation. At the value k = 1000, around 80% of the variance are learnt, and hence we can consider only the top 1000 features and still obtain good performance.

With k = 50 found in the previous sections, calculate the reconstruction residual MSE error when using LSI and NMF

Error value for LSI	$\|\|\mathbf{X} - \mathbf{U_k}\sum_{\mathbf{k}}\mathbf{V_k}\|\|_F^2$	1188.0911602927142
Error value for NMF	$\|\|\mathbf{X} - \mathbf{WH}\|\|_F^2$	1215.083164008801

The reconstruction residual MSE Error when Using LSI and NMF are given in the above table.
NMF calculates how well each document fits each topic, rather than assuming a document

has multiple topics, due to which it works better with shorter texts like tweets. Therefore, the reconstruction error of NMF is higher than that of LSI.

Question 5

Compare and contrast hard-margin and soft-margin linear SVMs:

- **Train two linear SVMs:**
- **Train one SVM with $\gamma = 1000$ (hard margin), another with $\gamma = 0.0001$ (soft margin).**
- **Plot the ROC curve, report the confusion matrix and calculate the accuracy, recall, precision and F-1 score of both SVM classifiers on the testing set. Which one performs better? What about for $\gamma = 100000$?**
- **What happens for the soft margin SVM? Why is the case? Analyze in terms of the confusion matrix.**
- **Does the ROC curve of the soft margin SVM look competitive? Please explain.**
- **Use cross-validation to choose γ (use average validation 3 accuracy to compare): Using a 5-fold cross-validation, find the best value of the parameter γ in the range $10^k| - 3 <= k <= 6, k \in Z$. Again, plot the ROC curve and report the confusion matrix and calculate the accuracy, recall precision and F1 score of this best SVM.**

In this question, Comparison is carried out between the Hard margin and Soft Margin Support vector Machines(SVM) with Linear kernals. Generally, they are chosen depending upon the problem and dataset involved, as stated below:

Code for Question 5.a:

```
#Import for Classifier metrics Calculation- from sklearn.metrics import confusion_matr
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_sc
def Classifier_Metrics(y_test,y_pred,name="",average='macro'):
    print("Accuracy score for %s: %f" %(name,accuracy_score(y_test,y_pred)))
    print("Recall score for %s: %f" % (name,recall_score(y_test,y_pred,average=average))
    print("Precision score for %s: %f" % (name,precision_score(y_test,y_pred,average=ave
    print("F-1 score for %s: %f" % (name,f1_score(y_test,y_pred,average=average)))

# Computing Hard and Soft Margin SVMs
from sklearn.svm import LinearSVC
from sklearn import svm
hardMargin_SVM=svm.SVC(kernel='linear',C=1000, random_state=42)

softMargin_SVM=svm.SVC(kernel='linear',C=0.0001, random_state=42)
```

```
# Normalise Train dataset and Test dataset to make them compatible for Binary classifi
True_labels_Train=train.root_label.values
True_labels_Test=test.root_label.values
X_True_Train=[]

# Making True Train and Test labels compatible for Binary Classification
for label in True_labels_Train:
  if label=='sports':
    X_True_Train.append(0)
  else:
    X_True_Train.append(1)
X_True_Test=[]

for label in True_labels_Test:
  if label=='sports':
    X_True_Test.append(0)
  else:
    X_True_Test.append(1)

X_hardSVM_pred = hardMargin_SVM.fit(x_LSI_train,X_True_Train).predict(X_LSI_test) # pred
X_softSVM_pred = softMargin_SVM.fit(x_LSI_train,X_True_Train).predict(X_LSI_test) # pred

Classifier_Metrics(X_True_Test,X_hardSVM_pred,name="Hard Margin SVM")
Classifier_Metrics(X_True_Test,X_softSVM_pred,name="Soft Margin SVM")

def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=p
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
      for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], fmt),horizontalalignment="center",color="white"
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
class_names=["Sports","Climate"]
hardSVM_cm = confusion_matrix(X_True_Test,X_hardSVM_pred) # Hard SVM Confusion matrix
plt.figure(); plot_confusion_matrix(hardSVM_cm, classes=class_names, title='Hard SVM Con
softSVM_cm = confusion_matrix(X_True_Test,X_softSVM_pred) # Soft SVM Confusion matrix
plt.figure(); plot_confusion_matrix(softSVM_cm, classes=class_names, title='Soft SVM Con
```

```
#ROC Curves
def ROC_curve(y_test,decision_function,name=""):
    fpr = dict();tpr = dict();roc_auc = dict()
    fpr, tpr, thresholds = roc_curve(y_test, decision_function)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',lw=lw, label='ROC curve (area = %0.4f)' % roc_
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0]); plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate'); plt.ylabel('True Positive Rate');
    plt.title('%s ROC curve' % name);plt.legend(loc="lower right")

ROC_curve(X_True_Test,hardMargin_SVM.decision_function(X_LSI_test),name="Hard Margin SVM
ROC_curve(X_True_Test,softMargin_SVM.decision_function(X_LSI_test),name="Soft Margin SVM

#Hard Margin- gamma=100000
HighHardMargin_SVM=svm.SVC(kernel='linear',C=100000, random_state=42)
X_HighhardSVM_pred=HighHardMargin_SVM.fit(x_LSI_train,X_True_Train).predict(X_LSI_test)
Classifier_Metrics(X_True_Test,X_HighhardSVM_pred,name="High Hard Margin SVM")
HighhardSVM_cm = confusion_matrix(X_True_Test,X_HighhardSVM_pred) # Hard SVM Confusion
plt.figure(); plot_confusion_matrix(HighhardSVM_cm, classes=class_names, title='High Har
ROC_curve(X_True_Test,HighHardMargin_SVM.decision_function(X_LSI_test),name="High Hard M
```

In both the cases, the following performance metrics are calculated and compared to make an inference: Confusion Matrix, Accuracy, Recall, Precision and F1-score.

Results for Hard Margin SVM:

- Accuracy score for Hard Margin SVM: 0.966265
- Recall score for Hard Margin SVM: 0.956311
- Precision score for Hard Margin SVM: 0.975248
- F-1 score for Hard Margin SVM: 0.965686

Results for Soft Margin SVM:

- Accuracy score for Soft Margin SVM: 0.496386
- Recall score for Soft Margin SVM: 1.000000
- Precision score for Soft Margin SVM: 0.496386
- F-1 score for Soft Margin SVM: 0.663446

Results for Hard Margin SVM with γ =100000:

- Accuracy score for Hard Margin SVM: 0.966265

- Recall score for Hard Margin SVM: 0.951456
- Precision score for Hard Margin SVM: 0.980000
- F-1 score for Hard Margin SVM: 0.965517

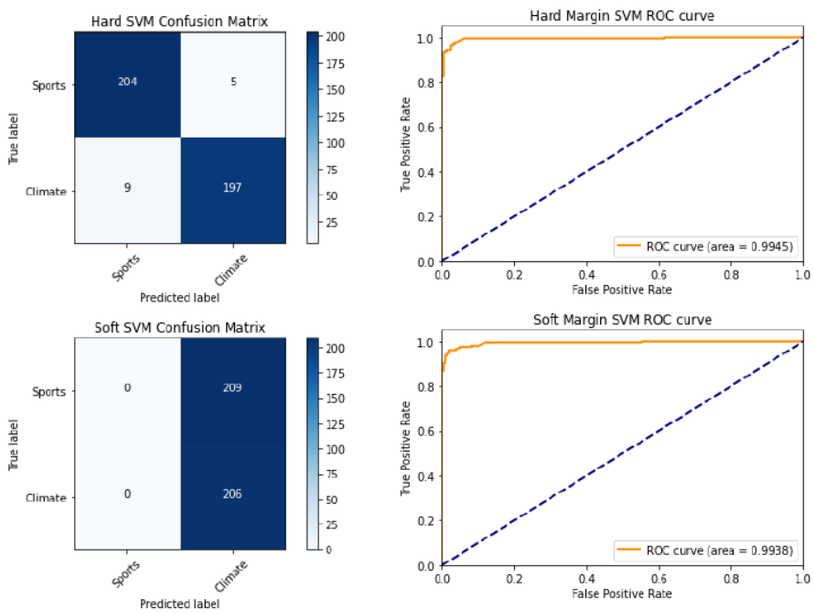

Figure 5: Hard Margin, Soft Margin- Confusion Matrices- ROC Curves

Hard margin SVM gives better results when compared with Soft margin SVM. When γ=100000, Performance remains saturated and doesn't increase due to the high value of Regularization strength.

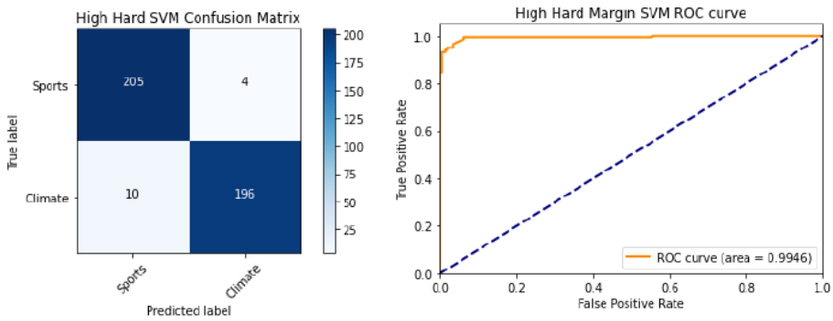

Figure 6: Confusion Matrices- ROC Curves- Hard Margin- γ =100000

Hard-Margin: When dataset under consideration is linearly separable, we can use SVM with a hard margin to have inferences without most misclassifications.

Soft-Margin: When having a linear boundary is not feasible with a given dataset and we wish the model to generalize well across samples, we prefer soft margin for our classifier jeopardizing some misclassifications.

Soft margin characterized by a low value γ tries to make the decision surface smooth and is prone to introducing high misclassification errors whereas a Hard margin with high value of γ aims at classifying all training examples correctly and does much more beyond forming a smooth flat decision surface and hence is very accurate here.

Confusion Matrix reflects the statement above. By observing the confusion matrix of Hard Margin, we find that the amount of True predictions is extremely high where the number of

False predictions is very less, whereas it is the converse in case of the Confusion matrix of Soft Margin SVM with high false predictions and very low True predictions.

Observing the Soft Margin's ROC curve, we find it contradictory to the other metrics such as confusion matrix, accuracy, precision and F1. All the other metrics with the exception of ROC, show that Soft Margin SVM has a very high misclassification rate for one of the classes and very low misclassification rate for the other class.

Further, the 5-fold Cross-Validation is performed to find the best value of γ (Regularization Strength) by sweeping through the range mentioned in the question.

The best γ is found to be 1000.

Code for Question 5.b:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
# svc = svm.SVC(random_state=42)
gamma=[0.001,0.01,0.1,1,10,100,1000,10000,100000,1000000]

best_gamma = 0.001
best_accuracy = 0
#Cross validation/scoring should happen on the training data
for g in gamma:
  clf = svm.SVC(kernel='linear', C=g, random_state=42)
  scores = cross_val_score(clf, x_LSI_train, X_True_Train, cv=5)
  #print(scores)
  print("Gamma value: %0.3f: %0.9f accuracy with a standard deviation of %0.2f" % (g, sc
  if scores.mean() > best_accuracy:
    best_accuracy = scores.mean()
    best_gamma = g

print(best_gamma)
BestHardMargin_SVM=svm.SVC(kernel='linear',C=best_gamma, random_state=42)   #Best-gamma=
X_BesthardSVM_pred=BestHardMargin_SVM.fit(x_LSI_train,X_True_Train).predict(X_LSI_test)
Classifier_Metrics(X_True_Test,X_BesthardSVM_pred,name="Best Hard Margin SVM")
Best_HighhardSVM_cm = confusion_matrix(X_True_Test,X_BesthardSVM_pred) # Hard SVM Confus
plt.figure(); plot_confusion_matrix(Best_HighhardSVM_cm, classes=class_names, title='Hig
ROC_curve(X_True_Test,BestHardMargin_SVM.decision_function(X_LSI_test),name="High Hard M
```

Accuracy, Recall precision, F1-score metrics obtained are reported below:
Results for SVM with γ =1000:

- Accuracy score for Hard Margin SVM: 0.966265
- Recall score for Hard Margin SVM: 0.956311
- Precision score for Hard Margin SVM: 0.975248
- F-1 score for Hard Margin SVM: 0.965686

ROC Curve and Confusion Matrix are plotted below:

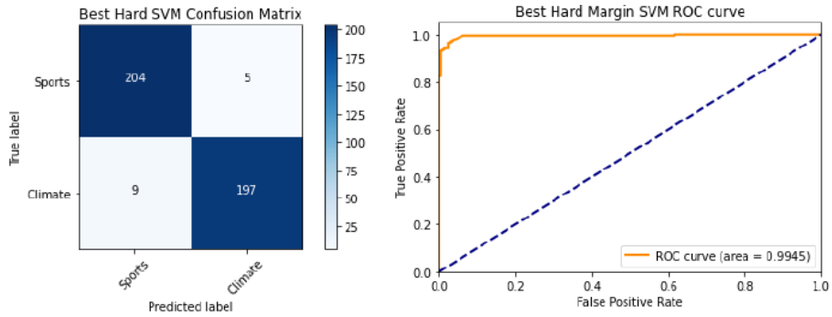

Figure 7: 5 fold- Cross Validation- Confusion Matrices- ROC Curves

Question 6

Evaluate a logistic classifier:

Train a logistic classifier without regularization (you may need to come up with some way to approximate this if you use sklearn.linear model.LogisticRegression)

In this question, using **Logistic Regression**, we train and compare performance metrics with and without the use of **L1,L2 regularization** parameters. In addition to this, we use cross validation to figure out the best regularization parameters.

Code for Question 6:

```
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(penalty = 'none', random_state = 42)
pred_lr = LR.fit(x_LSI_train,X_True_Train).predict(X_LSI_test)

print("Coefficients learned by logistic regression without regularization: ", LR.coef_)
Classifier_Metrics(X_True_Test, pred_lr, name="Logistic Regression without regularizatio
Logistics_Regression_cm = confusion_matrix(X_True_Test,pred_lr) # Hard SVM Confusion ma
plt.figure(); plot_confusion_matrix(Logistics_Regression_cm, classes=class_names, title=
ROC_curve(X_True_Test,LR.decision_function(X_LSI_test),name="Logistic Regression without
```

Plot the ROC curve and report the confusion matrix and calculate the accuracy, recall precision and F-1 score of this classifier on the testing set.

Comparison of Performance Results:

Without Regularization:

Results for Logistic Regression without Regularization:

- Accuracy score for Logistic Regression without Regularization: 0.966265
- Recall score for Logistic Regression without Regularization: 0.956311

- Precision score for Logistic Regression without Regularization: 0.975248
- F-1 score for Logistic Regression without Regularization: 0.965686

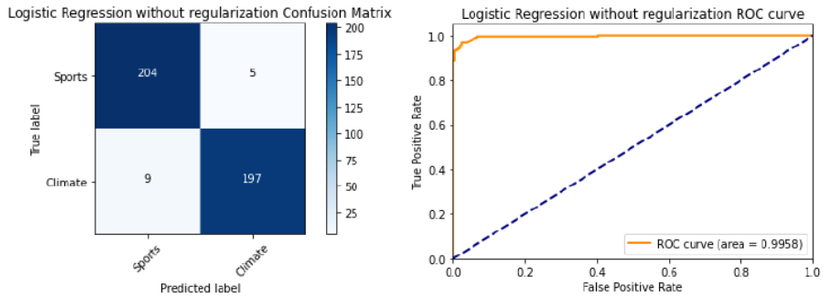

Figure 8: 5 fold- Cross Validation- Confusion Matrices- ROC Curves

Find the optimal regularization coefficient:

Using 5-fold cross-validation on the dimension-reduced-by-SVD training data, find the optimal regularization strength in the range $10^k | 4 <= k <= 4, k \in Z$ for logistic regression with L1 regularization and logistic regression with L2 regularization, respectively

Optimal Regularization Coefficient for L1 Regularization: k=100

Optimal Regularization Coefficient for L2 Regularization: k=1000

Code for the above Question:

```
#Using liblinear

l1reg=[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

#Need to choose Solvers which support both L1 and L2 regularization
#Saga, Liblinear solvers support both

l1bestk = 0
best_accuracy = 0
#Cross validation/scoring should happen on the test data
#Find the value of 'C'- Cfloat, default=1.0
#Inverse of regularization strength; must be a positive float. Like in support vector
print("L1 Regularization Scores")
l1_scores=[]
l1_learned_Coeff=[]
for l1 in l1reg:
  LR = LogisticRegression( penalty='l1',random_state = 42,solver='liblinear', C=l1)
  scores = cross_val_score(LR, X_LSI_test, X_True_Test, cv=5)
  l1_scores.append(scores.mean())
  #print(scores)
  print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std
  if scores.mean() > best_accuracy:
    best_accuracy = scores.mean()
```

```
    l1bestk = l1

print("Best k = ", l1bestk)

print("L2 Regularization Scores:")
l2bestk = 0
best_accuracy = 0
l2_scores=[]
#Cross validation/scoring should happen on the test data
#Find the value of 'C'- Cfloat, default=1.0
#Inverse of regularization strength; must be a positive float. Like in support vector
for l2 in l1reg:
  LR = LogisticRegression( penalty='l2',random_state = 42,solver='liblinear', C=l2)
  scores = cross_val_score(LR, X_LSI_test, X_True_Test, cv=5)
  l2_scores.append(scores.mean())
  #print(scores)
  print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std
  if scores.mean() > best_accuracy:
    best_accuracy = scores.mean()
    l2bestk = l2

print("Best k = ", l2bestk)
#Performance with L1 Regularization
print("Performance with L1 Regularization")
LR = LogisticRegression( penalty='l1',random_state = 42,solver='liblinear', C=l1bestk)
pred_lr = LR.fit(x_LSI_train,X_True_Train).predict(X_LSI_test)
Classifier_Metrics(X_True_Test, pred_lr, name="Logistic Regression with L1 regularizatio
Logistics_Regression_cm = confusion_matrix(X_True_Test,pred_lr) # Hard SVM Confusion ma
plt.figure(); plot_confusion_matrix(Logistics_Regression_cm, classes=class_names, title=
ROC_curve(X_True_Test,LR.decision_function(X_LSI_test),name="Logistic Regression with L1

#Performance with L2 Regularization
print("Performance with L2 Regularization")
LR = LogisticRegression( penalty='l2',random_state = 42,solver='liblinear', C=l2bestk)
pred_lr = LR.fit(x_LSI_train,X_True_Train).predict(X_LSI_test)
Classifier_Metrics(X_True_Test, pred_lr, name="Logistic Regression with L2 regularizatio
Logistics_Regression_cm = confusion_matrix(X_True_Test,pred_lr) # Hard SVM Confusion ma
plt.figure(); plot_confusion_matrix(Logistics_Regression_cm, classes=class_names, title=
ROC_curve(X_True_Test,LR.decision_function(X_LSI_test),name="Logistic Regression with L2

#Performance with no Regularization
print("Performance with L2 Regularization")
LR = LogisticRegression( penalty='none',random_state = 42)
pred_lr = LR.fit(x_LSI_train,X_True_Train).predict(X_LSI_test)
Classifier_Metrics(X_True_Test, pred_lr, name="Logistic Regression without regularizatio
Logistics_Regression_cm = confusion_matrix(X_True_Test,pred_lr) # Hard SVM Confusion ma
plt.figure(); plot_confusion_matrix(Logistics_Regression_cm, classes=class_names, title=
ROC_curve(X_True_Test,LR.decision_function(X_LSI_test),name="Logistic Regression without
```

With L1 Regularization:

Results for Logistic Regression with L1 Regularization:

- Accuracy score for Logistic Regression with L1 Regularization: 0.973494
- Recall score for Logistic Regression with L1 Regularization: 0.966019
- Precision score for Logistic Regression with L1 Regularization: 0.980296
- F-1 score for Logistic Regression with L1 Regularization: 0.973105

With L1 Regularization: With L2 Regularization:

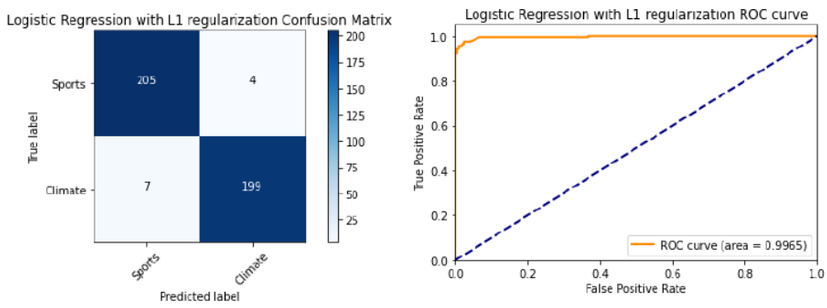

Figure 9: Logistic Regression with L1 Regularization- Confusion Matrices- ROC Curves

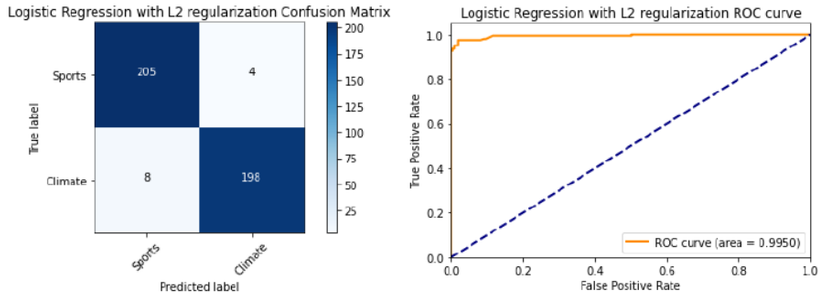

Figure 10: Logistic Regression with L2 Regularization- Confusion Matrices- ROC Curves

Results for Logistic Regression with L2 Regularization:

- Accuracy score for Logistic Regression with L2 Regularization: 0.971084
- Recall score for Logistic Regression with L2 Regularization: 0.961165
- Precision score for Logistic Regression with L2 Regularization: 0.980198
- F-1 score for Logistic Regression with L2 Regularization: 0.970588

Compare the performance (accuracy, precision, recall and F-1 score) of 3 logistic classifiers: w/o regularization, w/ L1 regularization and w/ L2 regularization (with the best parameters you found from the part above), using test data. How does the regularization parameter affect the test error? How are the learnt coefficients affected? Why might one be interested in each type of regularization?

Observing the Performance Metrics(listed above) from these cases, We see that Logistic Regression with L1 Regularization is having a slight increase in performance compared to the case without Regularization.

Choice of Regularization:

- **L1 regularization** is desired for feature selection. It discards features associated with zero weights leading to a simple and sparse model. The reason is L1/ Lasso Regularization tends to move all kinds of weights to 0 because of the sign(w) subgradient nature based on the $||w||$ relation. Also L1 regularization is more prone to return 0 coefficients when compared with L2 regularization with similar test accuracies.
- **L2 regularization**, in contrast to L1 regularization relies on w^2 as opposed to $||w||$, meaning it not only depends just on the sign but also the magnitude of the coefficients. Hence, L2 regularization is not recommended if feature selection is the main motive. Also, L2 regularization leads to models with diffuse weights. L2 regularization is preferred for decoupling the effects of correlated features. As a result of this process, it becomes susceptible to outliers and unstable at times.

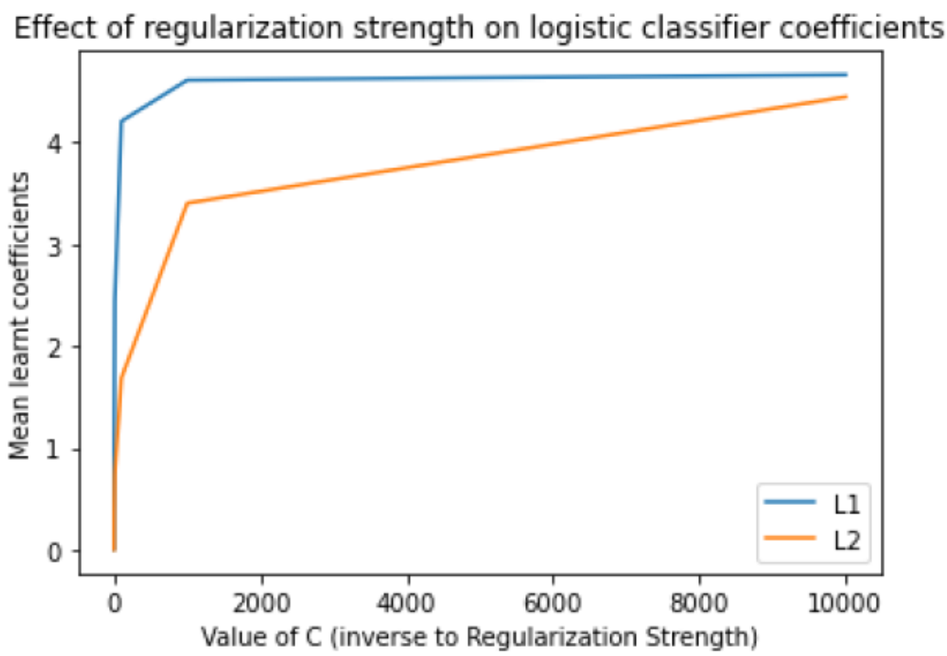

Figure 11: Effect of regularization strength on Logistic classifier coefficients on Mean Learnt Coefficients

Plots depicting the effect of regularization strength on Logistic classifier coefficients on Mean Learnt Coefficients and Mean Test Accuracy Could be found above.

- From these results, we can observe that as C/gamma increases -(which is inversely proportional to Regularization strength) the mean value of logistic regression coefficients along with test accuracy decreases.
- This is because, more weights in the learned model are set to 0.
- Again, this is due to the inherent formulations of L1 (Lasso) and L2 (Ridge) regularizations w^2 as opposed to $||w||$.

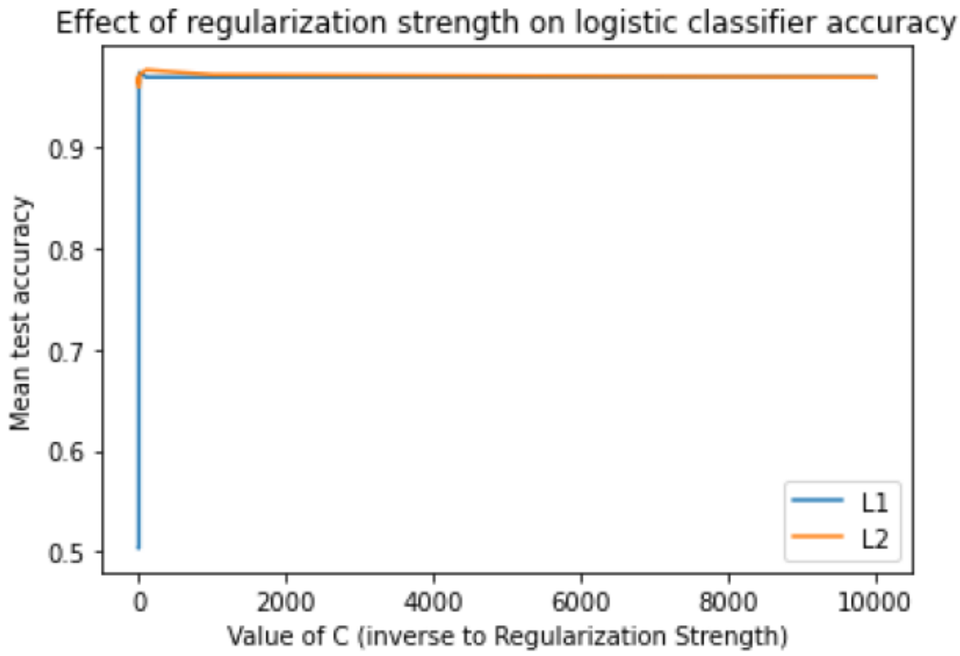

Figure 12: Effect of regularization strength on Logistic classifier coefficients on Mean Test Accuracy

- Throughout the training process, as the model starts to become more complex, important weights required for segregating features might be discarded. As a result of this, high regularization strengths inevitably might lead to poor test accuracies.
- Nominal finite values of Regularization strength will help the model to generalize well during inference, thereby circumventing the model from overfitting to the training samples.
- In addition to these effects, Regularization decreases variance and sensitivity to outliers and thereby makes the model more stable

Both Logistic Regression and Linear SVM are trying to classify data points using a linear decision boundary. What is the difference between their ways to find this boundary? Why do their performances differ? Is this difference statistically significant?

Both SVM and Logistic Regression tend to classify points using a linear decision boundary. Following are some of the key diffences between SVM and Logistic Regression:

- SVM is a geometric and deterministic algorithm which operates on the concept of Support Vectors. These support vectors are nothing but a subset of points closest to the decision boundary. SVM uses these support vectors to find an optimal separating hyperplane which maximizes the distance of the vectors to the margin.
- Logistic Regression is a statistical/probabilistic algorithm which intends to maximize the conditional likelihood to perform accurate classification. In other words, we turn to Logistic Regression, when we want to maximize the likelihood that a random data point gets classified correctly, this is called as Maximum Likelihood Estimation.
- In general, if the dataset under consideration is not linearly separable, Logistic Regression will fail, whereas SVM with a non-linear kernel (such as Radial Basis function

(RBF)) will still to find the decision boundary by transforming datasets into rich feature space, where the complex problems could still be mapped back to a linear fashion in the elevated hyper space.

- Statistically, While SVM tries to find the widest possible separating margin using non-linear kernels, Logistic Regression optimizes the log likelihood function, with probabilities modeled by sigmoid kind of functions.

Question 7

Evaluate and profile a Naive Bayes classifier: Train a GaussianNB classifier;

In this question we are asked to use Naive Bayes Classifier and record the corresponding Performance metrics. Code for this question is shown below:

```
from sklearn.naive_bayes import GaussianNB

GNB = GaussianNB()
GNB_pred_Y = GNB.fit(x_LSI_train,X_True_Train).predict(X_LSI_test)
Classifier_Metrics(X_True_Test, GNB_pred_Y, name="Naive Bayes Classifier")
GNB_pred_Y_cm = confusion_matrix(X_True_Test,GNB_pred_Y) # Hard SVM Confusion matrix
plt.figure()
plot_confusion_matrix(GNB_pred_Y_cm, classes=class_names, title='Naive Bayes Classifier
#Gaussian Naive Bayes Estimator does not have decision function attribute- cannot get
#So use Prediction probability estimate of test vectors for plotting ROC curve
ROC_curve(X_True_Test,GNB.predict_proba(X_LSI_test)[:,1],name="Naive Bayes Classifier RO
```

Plot the ROC curve and report the confusion matrix and calculate the accuracy, recall, precision and F-1 score of this classifier on the testing set.

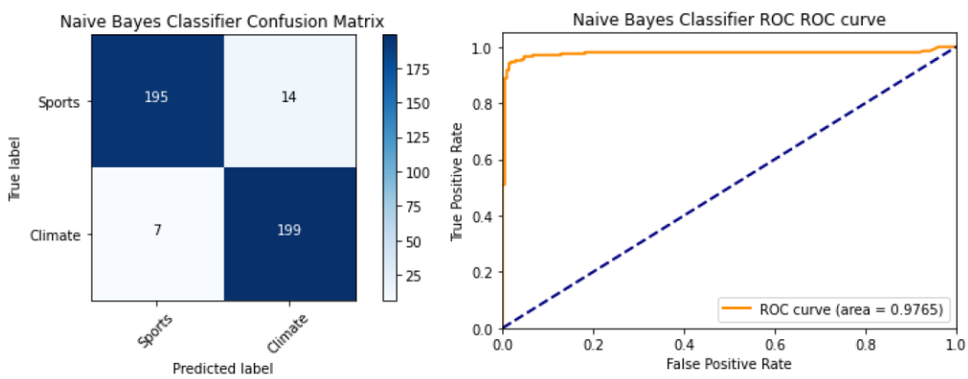

Figure 13: Naive Bayes Classifier- Confusion Matrix- ROC Curve

Results for Naive Bayes Classifier:

- Accuracy score for Naive Bayes Classifier: 0.949398
- Recall score for Naive Bayes Classifier: 0.966019
- Precision score for Naive Bayes Classifier: 0.934272

- F-1 score for Naive Bayes Classifier: 0.949881

Question 8

In this part, you will attempt to find the best model for binary classification.

- **Construct a Pipeline that performs feature extraction, dimensionality reduction and classification;**
- **The evaluation of each combination is performed with 5-fold cross-validation (use the average validation set accuracy across folds).**
- **In addition to any other hyperparameters you choose, your gridsearch must at least include.**
- **What are the 5 best combinations? Report their performances on the testing set.**

Following shows the components of our pipeline:

Module	Options
Loading Data	Cleaning vs not cleaning the data.
Feature Extraction	`min_df = 3` vs `5` while constructing the vocabulary; AND use Lemmatization vs Stemming vs Nothing as a compression module
Dimensionality Reduction	LSI ($k = [5, 50, 500]$) vs NMF ($k = [5, 50, 500]$)
Classifier	SVM with the best γ previously found vs Logistic Regression: L1 regularization vs L2 regularization, with the best regularization strength previously found vs `GaussianNB` Note: You can once again find the optimal hyperparameters for each classifier, but this is not *required*.
Other options	Use default

Pipeline Combination which resulted in Best accuracy according to this observation:

	Classifier	Reduce_dim	Analyzer	min_df	Validation Score	Cleaning	Test Score
1	SVM	Truncated SVD 500	Stemmed	3	0.969206	clean	0.980723
2	Logistic Regression L1	Truncated SVD 500	Stemmed	3	0.970418	unclean	0.980621
3	SVM	Truncated SVD 500	Lemmatized	5	0.968005	clean	0.975904
4	Logistic Regression L1	Truncated SVD 500	Lemmatized	3	0.967406	unclean	0.975891
5	SVM	Truncated SVD 500	Stemmed	5	0.96921	unclean	0.973494

Here, the question asks to find the optimal combination of all the feature extraction parameters, dimensionality reduction approaches and Classification Algorithms utilizing Pipeline feature to construct the entire Pipeline with 5-fold Grid Search Cross validation available in Scikit-learn.

Question 9

In this part, we aim to learn classifiers on the documents belonging to unique classes in the column leaf label.

Perform Naive Bayes classification and multiclass SVM classification (with both One VS One and One VS the rest methods described above) and report the confusion matrix and calculate the accuracy, recall, precision and F-1 score of your classifiers. How did you resolve the class imbalance issue in the One VS the rest model?

In the confusion matrix you should have an 8 × 8 matrix where 8 is the number of unique labels in the column leaf label. Please make sure that the order of these labels is as follows: map_row_to_class = 0:"chess", 1:"cricket", 2:"soccer", 3:"football", ,→ 4:"%22forest%20fire%22", 5:"flood", 6:"earthquake", 7:"drought" **Do you observe any structure in the confusion matrix? Are there distinct visible blocks on the major diagonal? What does this mean?**

Observation

- We observe Diagonal values taking the highest values across the Confusion Matrix
- Existence of such visible individual blocks would mean that we have a comparatively good and high magnitude of True Predictions or correct classifications
- However, we could do better by using the merging labels and subsampling technique.

For SVM, Two possible approaches exist for Multiclass Classification

- In one vs one classification for n class labels, the entire dataset is partitioned into n binary datasets. Following this, classifiers are trained on the dataset, using either the majority vote or based on the confidence levels. As we compare one on one between all classes $n * (n-1)/2$ models are created.
- In one vs rest classification for n class labels, the entire dataset is partitioned into n datasets, with each dataset containing a positive class and all other classes assigned as negative. Then N number of classifiers are trained on the dataset.
- Imbalance - The Dataset partitioning scheme used in one vs rest followed by classification is the reason for dataset imbalance particularly during the one vs rest multiclass classification. This is because in the one vs rest classification, the number of elements not belonging to the class far outweigh the ones that belong to it. To avoid this, we can set $class_weight =' balanced'$. This provides a weight to the class that is inversely proportional to their respective frequencies.
- The accuracy of one vs one is better than the one vs rest. Though above methods were used, class imbalances could have played a factor in this.

Code for this question:

```
mtrain, mtest= train_test_split(data[["full_text","leaf_label"]], test_size=0.2)
print("Number of Training Samples:", len(mtrain))
print("Number of Testing Samples:", len(mtest))
print(mtrain)
#Multiclass label
mtrain['clean'] = mtrain['full_text'].apply(lambda x:preprocess(x))
mtest['clean'] = mtest['full_text'].apply(lambda x:preprocess(x))
print(mtrain)
#Multiclass Classification Count Vectorizer and TF_IDF Transformer

word_count_vec_mtrain = count_vec.fit_transform(mtrain['clean'])
word_count_vec_mtrain_tf = tfidf_tranformer.fit_transform(word_count_vec_mtrain)
word_count_array_mtrain = word_count_vec_mtrain_tf.toarray()
word_counts_mtrain = pd.DataFrame(data=word_count_array_mtrain, columns = count_vec.get_

print(word_counts_mtrain.shape)

word_count_vec_mtest = count_vec.transform(mtest['clean'])
word_count_vec_mtest_tf = tfidf_tranformer.transform(word_count_vec_mtest)
word_count_array_mtest = word_count_vec_mtest_tf.toarray()
word_counts_mtest = pd.DataFrame(data=word_count_array_mtest, columns = count_vec.get_fe
print(word_counts_mtest.shape)

#Multiclass Classification- Best gamma looks like 100

#Multiclass Classification

from sklearn.model_selection import GridSearchCV
svm_ovo = OneVsOneClassifier(SVC(kernel = 'linear', random_state=42, C=100))
param_grid = {'estimator__C': [0.001,0.01,0.1,1,10,100,1000,10000,100000,1000000]}
#grid_svm_mc = GridSearchCV(svm_ovo, param_grid, cv=5, scoring='accuracy')
pred_svm_mc=svm_ovo.fit(x_LSI_mtrain, MCN_labels_Train).predict(X_LSI_mtest)
#pred_svm_mc = svm_ovo.best_estimator_.predict(X_LSI_mtest)

#print(svm_ovo.best_estimator_)
Classifier_Metrics_Multiclass(MCN_labels_Test, pred_svm_mc, name="One vs One SVM")
pred_svm_mc_cm = confusion_matrix(MCN_labels_Test,pred_svm_mc) # Hard SVM Confusion mat
plt.figure()
plot_confusion_matrix(pred_svm_mc_cm, classes=Mclass_names, title='One vs One SVM Confus

#Multiclass Classification
from sklearn.model_selection import GridSearchCV
svm_ovr = OneVsRestClassifier(SVC(kernel = 'linear', random_state=42, class_weight = 'ba
param_grid = {'estimator__C': [0.001,0.01,0.1,1,10,100,1000,10000,100000,1000000]}
#grid_svm_mc = GridSearchCV(svm_ovr, param_grid, cv=5, scoring='accuracy')
pred_svm_mc=svm_ovr.fit(x_LSI_mtrain, MCN_labels_Train).predict(X_LSI_mtest)
#pred_svm_mc = svm_ovr.best_estimator_.predict(X_LSI_mtest)
```

```
#print(svm_ovr.best_estimator_)
Classifier_Metrics_Multiclass(MCN_labels_Test, pred_svm_mc, name="One vs Rest SVM")
pred_svm_mc_cm = confusion_matrix(MCN_labels_Test,pred_svm_mc) # Hard SVM Confusion mat
plt.figure()
plot_confusion_matrix(pred_svm_mc_cm, classes=Mclass_names, title='One vs Rest SVM Confu
```

Performance Metrics for One vs One SVM:

- Accuracy score for One vs One SVM: 0.927711
- Recall score for One vs One SVM: 0.929219
- Precision score for One vs One SVM: 0.927608
- F-1 score for One vs One SVM: 0.927430

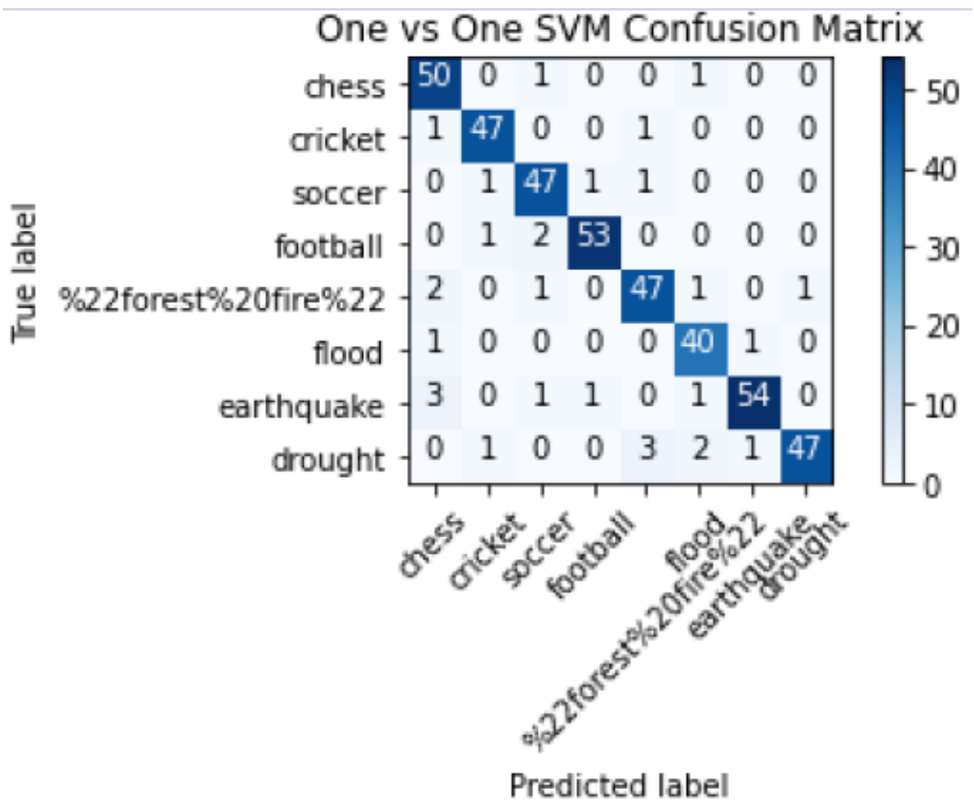

Figure 14: One vs One SVM- Confusion Matrix

Performance Metrics for One vs Rest SVM:

- Accuracy score for One vs Rest SVM: 0.908434
- Recall score for One vs Rest SVM: 0.911063
- Precision score for One vs Rest SVM: 0.910491
- F-1 score for One vs Rest SVM: 0.908107

In addition, answer the following questions

Based on your observation from the previous part, suggest a subset of labels that should be merged into a new larger label and recompute the accuracy and

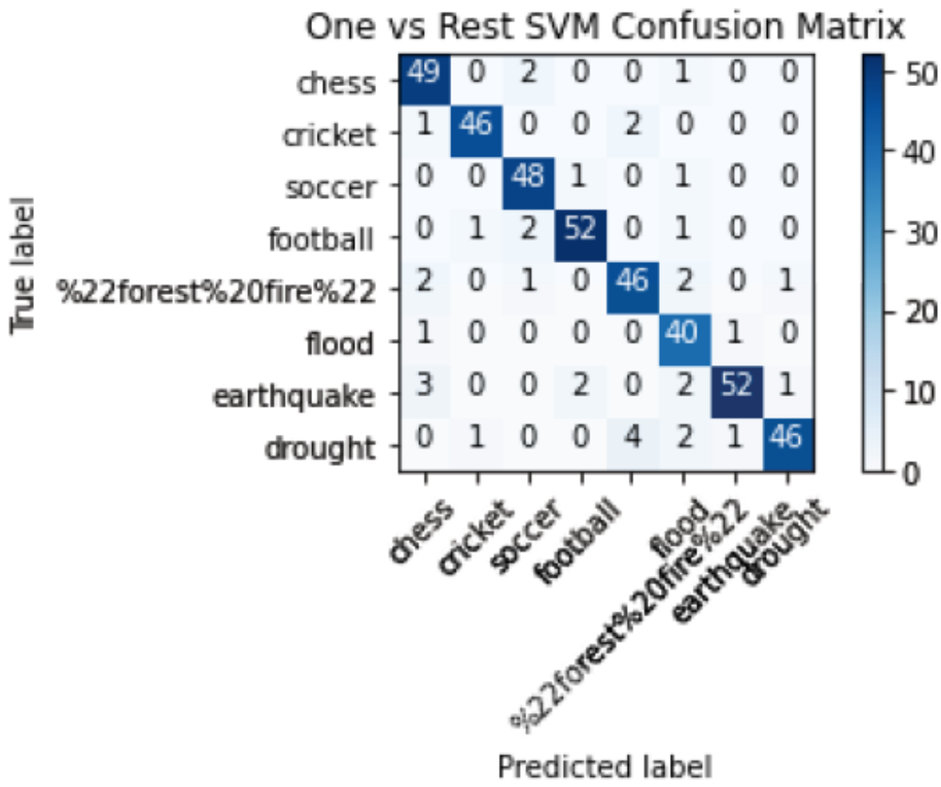

Figure 15: One vs Rest SVM- Confusion Matrix

plot the confusion matrix. How did the accuracy change in One VS One and One VS the rest?

Reasoning

- Data partitioning and classification inherent to one vs rest scheme is the reason for the imbalance.
- This imbalance and decreased accuracy due to imbalance could be effectively improved by reducing the number of negative clusters that each classifier sees during classification
- One way to do this is to merge semantically related labels into a single class and performing classification on top of them.

Analysis of Results

- By analyzing results, we could understand the technique of 'Merging Semantic Labels' has improved the performance metrics both in the case of One vs One and One vs Rest SVMs
- Due to the reduced number of classes involved in classification, the dimensions of the Confusion matrix is reduced.
- Nevertheless, we could observe the improved classification performance through the Confusion matrix as well

Code for the above question:

```
#Multiclass Classification

#Merging Semantic labels
MCN_labels_Train=[]

#Making True Train and Test labels compatible for Binary Classification
for label in MC_labels_Train:
  if label == 'chess' or label == 'cricket':
    MCN_labels_Train.append(0)
  elif label == 'soccer' or label== 'football':
    MCN_labels_Train.append(1)

  elif label == 'flood' or label== 'drought':
    MCN_labels_Train.append(2)
  elif label == 'earthquake':
    MCN_labels_Train.append(3)
  else:
    MCN_labels_Train.append(3)

MCN_labels_Test=[]
#Making True Train and Test labels compatible for Binary Classification
for label in MC_labels_Test:
  if label == 'chess' or label == 'cricket':
    MCN_labels_Test.append(0)
  elif label == 'soccer' or label== 'football':
    MCN_labels_Test.append(1)
  elif label == 'flood' or label== 'drought':
    MCN_labels_Test.append(2)
  elif label == 'earthquake':
    MCN_labels_Test.append(3)
  else:
    MCN_labels_Test.append(3)
Mclass_names = ["chess/cricket", "soccer/football", "%22forest%20fire%22 / earthquake",

from sklearn.model_selection import GridSearchCV
svm_ovo = OneVsOneClassifier(SVC(kernel = 'linear', random_state=42, C=100))
param_grid = {'estimator__C': [0.001,0.01,0.1,1,10,100,1000,10000,100000,1000000]}
#grid_svm_mc = GridSearchCV(svm_ovo, param_grid, cv=5, scoring='accuracy')
pred_svm_mc=svm_ovo.fit(x_LSI_mtrain, MCN_labels_Train).predict(X_LSI_mtest)
#pred_svm_mc = svm_ovo.best_estimator_.predict(X_LSI_mtest)

#print(svm_ovo.best_estimator_)
Classifier_Metrics_Multiclass(MCN_labels_Test, pred_svm_mc, name="One vs One SVM")
pred_svm_mc_cm = confusion_matrix(MCN_labels_Test,pred_svm_mc) # Hard SVM Confusion mat
plt.figure()
plot_confusion_matrix(pred_svm_mc_cm, classes=Mclass_names, title='One vs One SVM Confus

#Multiclass Classification
```

```
from sklearn.model_selection import GridSearchCV
svm_ovr = OneVsRestClassifier(SVC(kernel = 'linear', random_state=42, class_weight = 'ba
param_grid = {'estimator__C': [0.001,0.01,0.1,1,10,100,1000,10000,100000,1000000]}
#grid_svm_mc = GridSearchCV(svm_ovr, param_grid, cv=5, scoring='accuracy')
pred_svm_mc=svm_ovr.fit(x_LSI_mtrain, MCN_labels_Train).predict(X_LSI_mtest)
#pred_svm_mc = svm_ovr.best_estimator_.predict(X_LSI_mtest)

#print(svm_ovr.best_estimator_)
Classifier_Metrics_Multiclass(MCN_labels_Test, pred_svm_mc, name="One vs Rest SVM")
pred_svm_mc_cm = confusion_matrix(MCN_labels_Test,pred_svm_mc) # Hard SVM Confusion mat
plt.figure()
plot_confusion_matrix(pred_svm_mc_cm, classes=Mclass_names, title='One vs Rest SVM Confu
```

Performance metrics for One vs One SVM with Semantic Merging of Labels:

- Accuracy score for One vs One SVM: 0.944578
- Recall score for One vs One SVM: 0.940632
- Precision score for One vs One SVM: 0.949097
- F-1 score for One vs One SVM: 0.943800

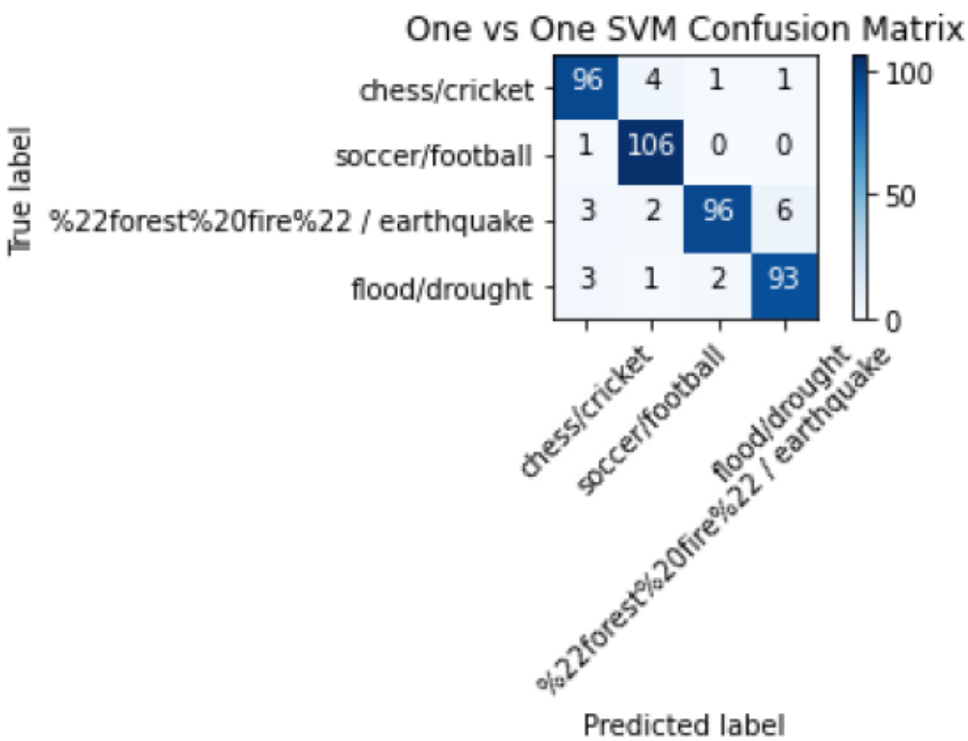

Figure 16: One vs One SVM-Merging Semantic Labels Confusion Matrix

Performance metrics for One vs Rest SVM with Semantic Merging of Labels:

- Accuracy score for One vs Rest SVM: 0.937349
- Recall score for One vs Rest SVM: 0.932450
- Precision score for One vs Rest SVM: 0.929175

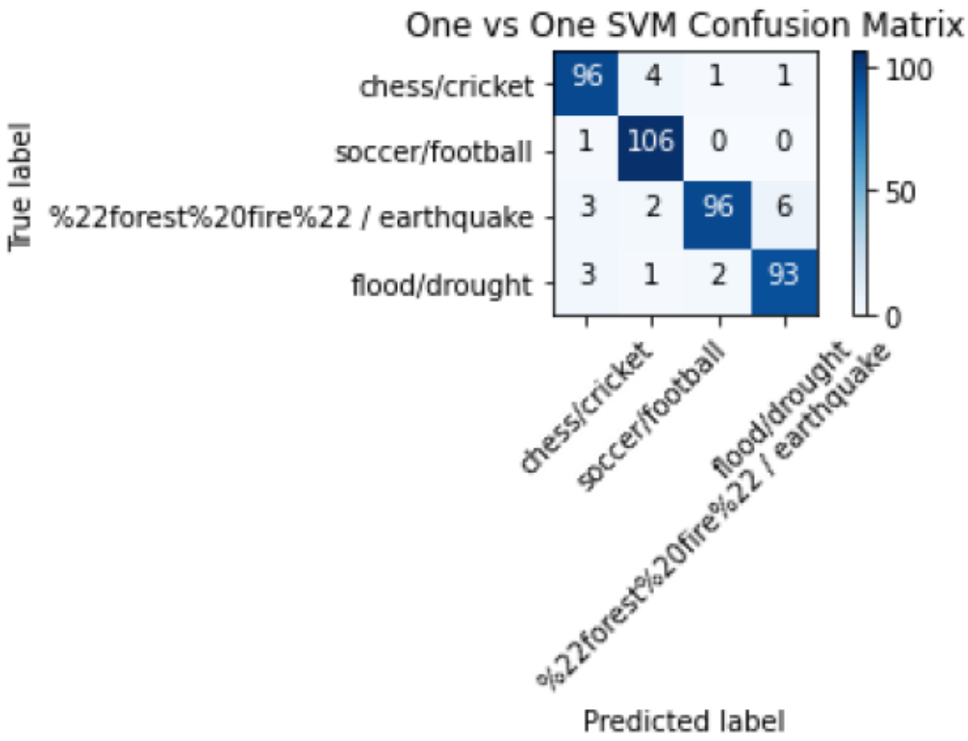

Figure 17: One vs One SVM-Merging Semantic Labels Confusion Matrix

- F-1 score for One vs Rest SVM: 0.929865

Does class imbalance impact the performance of the classification once some classes are merged? Provide a resolution for the class imbalance and recompute the accuracy and plot the confusion matrix in One VS One and One VS the rest?

Solution to Imbalance

- We incorporate the solution of Subsampling as part of our Classification pipeline to mitigate the effect of imbalance during classification. SMOTEEN method combines the SMOTE ability to generate synthetic examples for minority class and ENN ability to delete some observations from both classes that are identified as having different class between the observation's class and its K-nearest neighbor majority class. This way it combines undersampling and oversampling of the data.
- We observe that the results after applying the said method, there is an improvement in Performance Metrics.
- This confirms that we could be approaching in the right direction.

Code for the above question:

```
#Multiclass Classification

#Subsampling Techniques
#Multiclass Classification

from imblearn.combine import SMOTEENN
from imblearn.pipeline import Pipeline
```

```
from sklearn.datasets import make_classification
from imblearn.under_sampling import EditedNearestNeighbours

#Merging Semantic labels
MCN_labels_Train=[]

#Making True Train and Test labels compatible for Binary Classification
for label in MC_labels_Train:
  if label == 'chess' or label == 'cricket':
    MCN_labels_Train.append(0)
  elif label == 'soccer' or label== 'football':
    MCN_labels_Train.append(1)

  elif label == 'flood' or label== 'drought':
    MCN_labels_Train.append(2)
  elif label == 'earthquake':
    MCN_labels_Train.append(3)
  else:
    MCN_labels_Train.append(3)

MCN_labels_Test=[]
#Making True Train and Test labels compatible for Binary Classification
for label in MC_labels_Test:
  if label == 'chess' or label == 'cricket':
    MCN_labels_Test.append(0)
  elif label == 'soccer' or label== 'football':
    MCN_labels_Test.append(1)
  elif label == 'flood' or label== 'drought':
    MCN_labels_Test.append(2)
  elif label == 'earthquake':
    MCN_labels_Test.append(3)
  else:
    MCN_labels_Test.append(3)
Mclass_names = ["chess/cricket", "soccer/football", "%22forest%20fire%22 / earthquake",

from sklearn.model_selection import GridSearchCV
svm_ovo = OneVsOneClassifier(SVC(kernel = 'linear', random_state=42, C=100))
param_grid = {'estimator__C': [0.001,0.01,0.1,1,10,100,1000,10000,100000,1000000]}
#grid_svm_mc = GridSearchCV(svm_ovo, param_grid, cv=5, scoring='accuracy')
pred_svm_mc=svm_ovo.fit(x_LSI_mtrain, MCN_labels_Train).predict(X_LSI_mtest)
#pred_svm_mc = svm_ovo.best_estimator_.predict(X_LSI_mtest)

#print(svm_ovo.best_estimator_)
Classifier_Metrics_Multiclass(MCN_labels_Test, pred_svm_mc, name="One vs One SVM")
pred_svm_mc_cm = confusion_matrix(MCN_labels_Test,pred_svm_mc) # Hard SVM Confusion mat
plt.figure()
plot_confusion_matrix(pred_svm_mc_cm, classes=Mclass_names, title='One vs One SVM Confus
```

```
#Multiclass Classification
from sklearn.model_selection import GridSearchCV

#Introduce Subsampling in Pipeline
pipe = Pipeline([('sampl', SMOTEENN(sampling_strategy='all', random_state=42)),
                 ('clf', SVC(kernel = 'linear', random_state=42, C=100))])

svm_ovr = OneVsRestClassifier(pipe)
param_grid = {'estimator__C': [0.001,0.01,0.1,1,10,100,1000,10000,100000,1000000]}
#grid_svm_mc = GridSearchCV(svm_ovr, param_grid, cv=5, scoring='accuracy')
pred_svm_mc=svm_ovr.fit(x_LSI_mtrain, MCN_labels_Train).predict(X_LSI_mtest)
#pred_svm_mc = svm_ovr.best_estimator_.predict(X_LSI_mtest)

#print(svm_ovr.best_estimator_)
Classifier_Metrics_Multiclass(MCN_labels_Test, pred_svm_mc, name="One vs Rest SVM")
pred_svm_mc_cm = confusion_matrix(MCN_labels_Test,pred_svm_mc) # Hard SVM Confusion mat
plt.figure()
plot_confusion_matrix(pred_svm_mc_cm, classes=Mclass_names, title='One vs Rest SVM Confu


# This pipeline will resample the data and
# pass the output to MultinomialNB


# OVR will transform the `y` as you know and
# then pass single label data to different copies of pipe
# multiple times (as many labels in data)
```

Performance metrics for One vs One SVM -subsampling:

- Accuracy score for One vs One SVM: 0.944578
- Recall score for One vs One SVM: 0.944745
- Precision score for One vs One SVM: 0.944643
- F-1 score for One vs One SVM: 0.944168

Performance metrics for One vs Rest SVM -subsampling:

- Accuracy score for One vs Rest SVM: 0.942169
- Recall score for One vs Rest SVM: 0.942597
- Precision score for One vs Rest SVM: 0.942813
- F-1 score for One vs Rest SVM: 0.941801

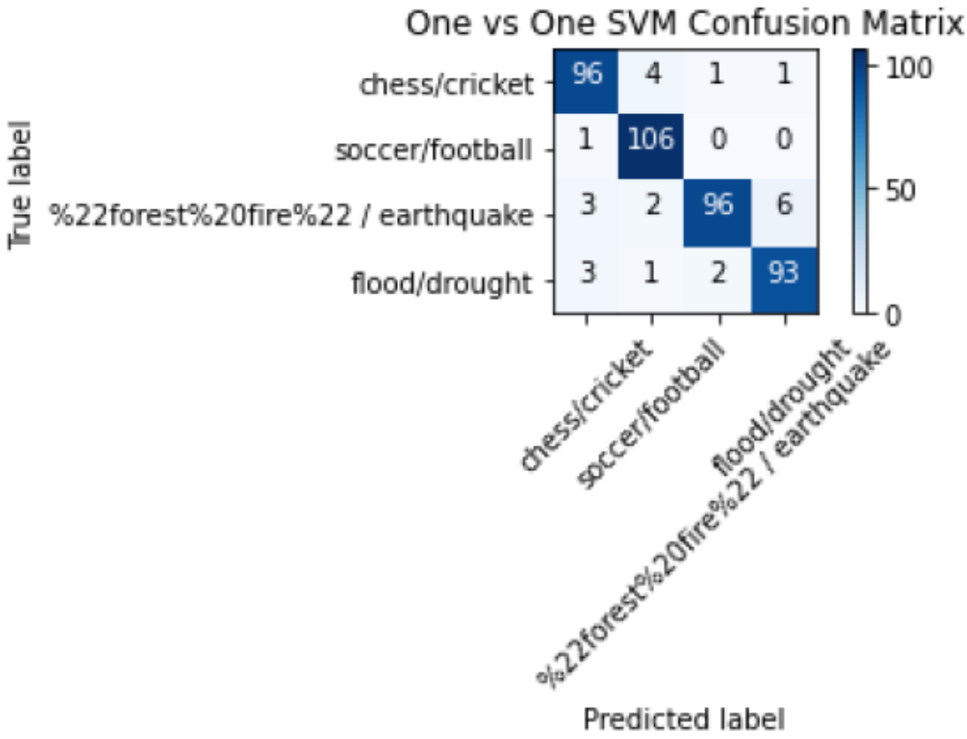

Figure 18: One vs One SVM-Subsampling- Confusion Matrix

Word Embedding

Question 10

Why are GLoVE embeddings trained on the ratio of co-occurrence probabilities rather than the probabilities themselves?

GloVE does not rely just on local context information of words, but incorporates word co-occurrence (global statistics) to obtain word vectors. The motivation behind using ratio of co-occurrence probabilities is context words which are nearer to a given word are more important as meaningful context than distant words. The words having ratio nearly equal to 1 either have good co-occurrence probability with both the words or none, hence causing no impact in learning the differentiation between given words. This proves ratio of co-occurrence probability is to be the starting point for learning word representations.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 20: Glove co-occurrence probability table

Consider 2 words i and j that have similar characteristics such as i = ice and j = steam. The relationship of these words can be examined by studying the ratio of their co-occurrence probabilities with various probe words, k. For words k related to ice and not related to steam, we expect P_{ik}/P_{jk} to be large, and for words k related to steam but not related to ice, the ratio will be small. For words that are neither related to i and j or related to both, the

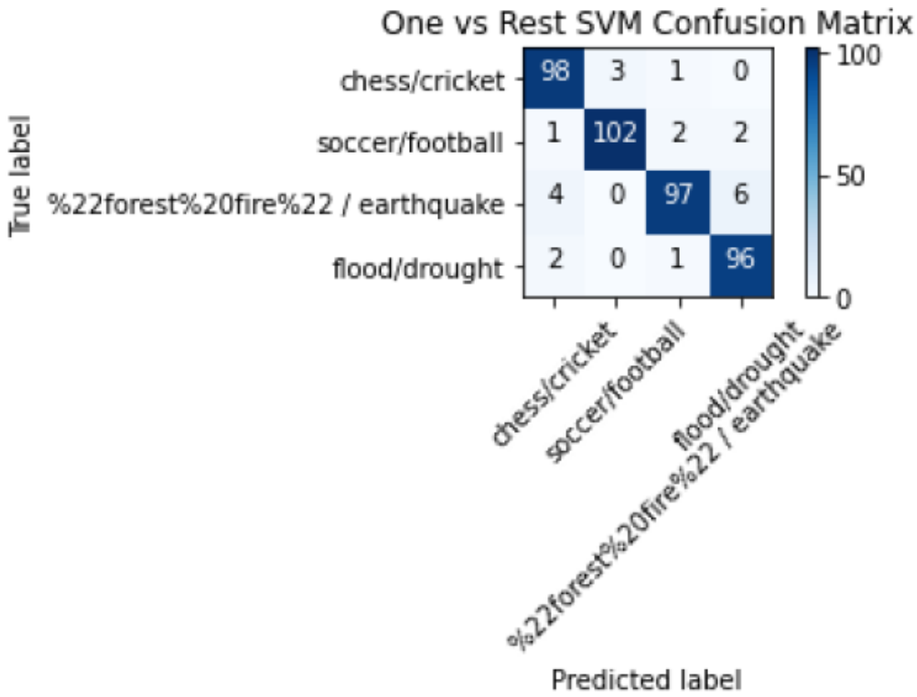

Figure 19: One vs Rest SVM-Subsampling- Confusion Matrix

ratio will be close to 1. The above figure shows these probabilities and their ratios for a large corpus, and the numbers confirm these expectations. Compared to the raw probabilities, the ratio is better able to distinguish relevant words (solid and gas) from irrelevant words (water and fashion) and it is also better able to discriminate between the two relevant words.

In the two sentences: "James is running in the park." and "James is running for the presidency.", would GLoVE embeddings return the same vector for the word running in both cases? Why or why not?
The GLoVE embeddings return the same vector for the word running in both cases. This is an instance of polysemy. While training, the corpus will have data coming in from both the meanings of the word "running" and hence the GLoVE vector that gets generated will be a linear superposition of all the various meanings of the word. From the experiments we noticed that the word is closer to the 1st meaning of running than to the 2^{nd}. The top closest words were ['ran', 'run', 'coming', 'going'].

What do you expect for the values of $||GLoVE["queen"]-GLoVE["king"]-GLoVE["wife"]+GLoVE["husband"]||_2$, $||GLoVE["queen"]-GLoVE["king"]||_2$ **and** $||GLoVE["wife"]-GLoVE["husband"]||_2$**. Compare the values**
The value of $||GLoVE["queen"]-GLoVE["king"]-GLoVE["wife"]+GLoVE["husband"]||_2$ is expected to be closer to 0.
The values for $||GLoVE["queen"]-GLoVE["king"]||_2$ and $||GLoVE["wife"]-GLoVE["husband"]||_2$ should be close to each other as there is a similar relationship between the 2 words. This can be explained as queen:king = wife:husband. Hence the difference in the vector representation of queen and king or wife and husband will be same as than of woman and man and hence contains the information regarding the gender change.

Given a word, would you rather stem or lemmatize the word before mapping it to its GLoVE embedding?
Stemming removes the last few characters of the words with the aim of simplifying the words to their base forms, without taking context into considerations. This can lead to words with-

out any meaning or correct spellings. Lemmatization will reduce the word to their root forms i.e., lemma after considering the context, semantics, parts of speech, etc.
Thus, it is better to choose lemmatization over stemming if necessary. However, it is mentioned in the paper (1) that manual lemmatization is not necessary, especially when using powerful contextualised embeddings. Their experiments showed that this is indeed true for languages with simple morphology (like English).

Reference:
(1) Kutuzov, Andrey, and Elizaveta Kuzmenko. "To Lemmatize or Not to Lemmatize: How Word Normalisation Affects ELMo Performance in Word Sense Disambiguation." DL4NLP 2019. Proceedings of the First NLPL Workshop on Deep Learning for Natural Language Processing, 30 September, 2019, University of Turku, Turku, Finland. No. 163. Linköping University Electronic Press, 2019.

Question 11

For the binary classification task distinguishing the "sports" class and "climate" class: (a) Describe a feature engineering process that uses GLoVE word embeddings to represent each document. You have to abide by the following rules:

- **A representation of a text segment needs to have a vector dimension that CANNOT exceed the dimension of the GLoVE embedding used per word of the segment.**
- **You cannot use TF-IDF scores (or any measure that requires looking at the complete dataset) as a pre-processing routine.**
- **Important: In this section, feel free to use raw features from any column in the original data file not just full text. The column keywords might be useful.**
- **To aggregate these words into a single vector consider normalizing the final vectors.**

The feature engineering process that uses GLoVE word embeddings to represent each document is described below. We followed all the rules mentioned in the question.

- We used a keyed vector model from Gensim to store the pre-trained word embeddings as word vectors. We represented the embeddings as a dictionary structure that maps keys (words) to their embedding vectors.
- We followed the same cleaning procedure as Question 3 where we removed the numbers, stopwords, punctuations and performed lemmatization.
- We converted the training documents into GLoVE features. We took a weighted average of the mean word embeddings of full-text, summary and key-words. For each of these we took the average across their word embeddings. We converted each text into a vector of length 300 yielding a matrix of size (1657, 300) for the training data and

(415, 300) for the test data. As we had taken the average, the data is ensured to be normalized.

(b) Select a classifier model, train and evaluate it with your GLoVE-based feature.
We used Linear SVM as the Classifier to train and evaluated. We had assumed $\gamma = 1000$ and random state = 42 and the results were poor. So we performed 5 fold validation to find the best γ of 10. These are the results:
Accuracy score for SVM: 0.944578
Recall score for SVM: 0.922330
Precision score for SVM: 0.964467
F-1 score for SVM: 0.942928

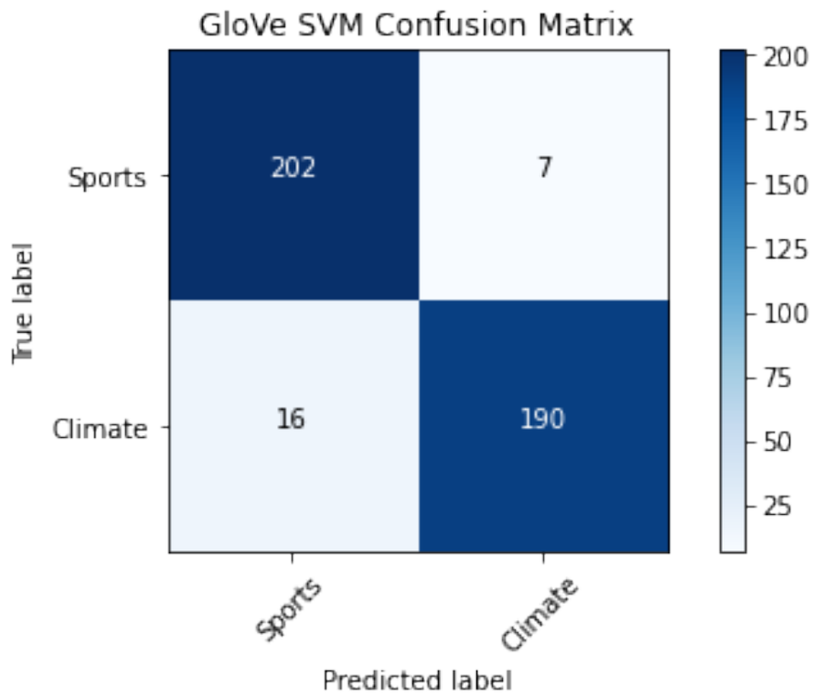

Figure 21: GLoVE SVM Confusion Matrix

Question 12

Plot the relationship between the dimension of the pre-trained GLoVE embedding and the resulting accuracy of the model in the classification task. Describe the observed trend. Is this trend expected? Why or why not? In this part use the different sets of GLoVE vectors from the link

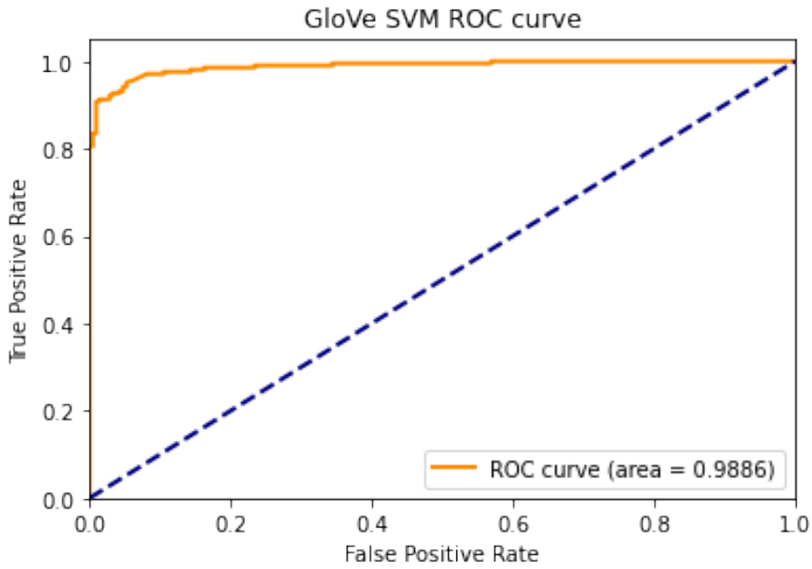

Figure 22: GLoVE SVM ROC Curve

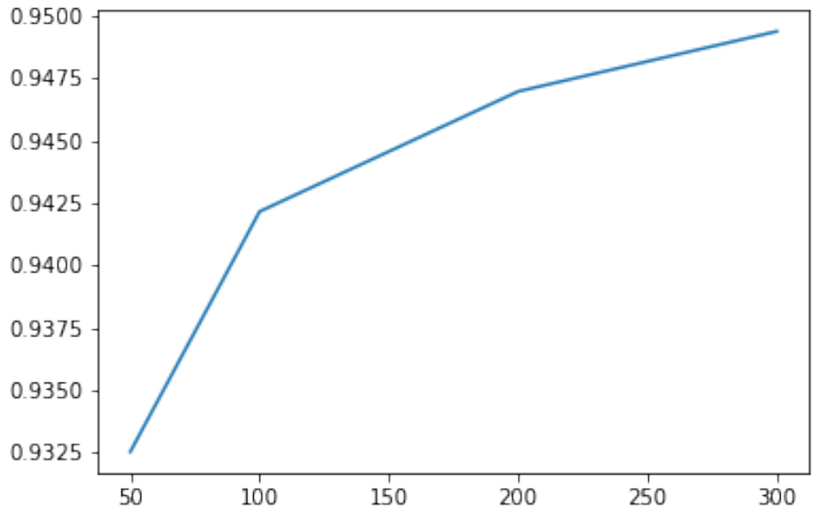

Figure 23: Plot of test accuracy vs GLoVE Embedding dimension for SVM Classifier

We retrained the linear SVM classifier in Question 11 ($\gamma = 100$) for various dimensions of word embeddings, namely 50, 100, 200, 300. Fig. 23 shows the plot for test accuracy vs. dimension of pre-trained GLoVE embedding.
We see that as the dimension of pre-trained GLoVE embedding increases, the test accuracy increases. This is expected because larger dimensions of GLoVE embedding capture much more semantic and complex information about distributed representation of words and co-occurrence probability ratios between target and context words in the latent space, which in turn provides more distinguishing features for the classifier to work on.

Question 13

Compare and contrast the two visualizations. Are there clusters formed in either or both of the plots? We will pursue the clustering aspect further in the next project.

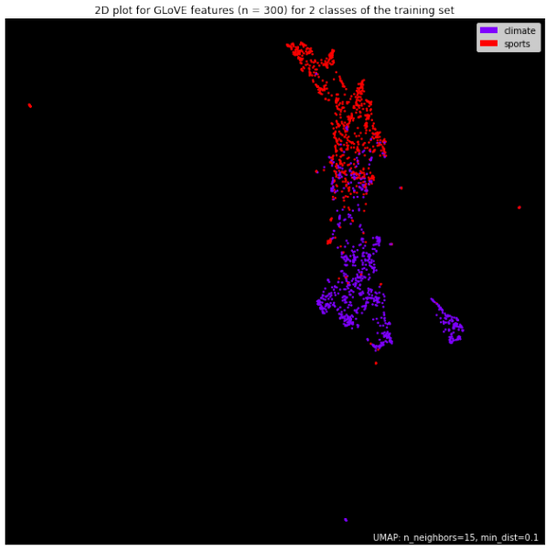

Figure 24: 2D Plot for GLoVE Training set

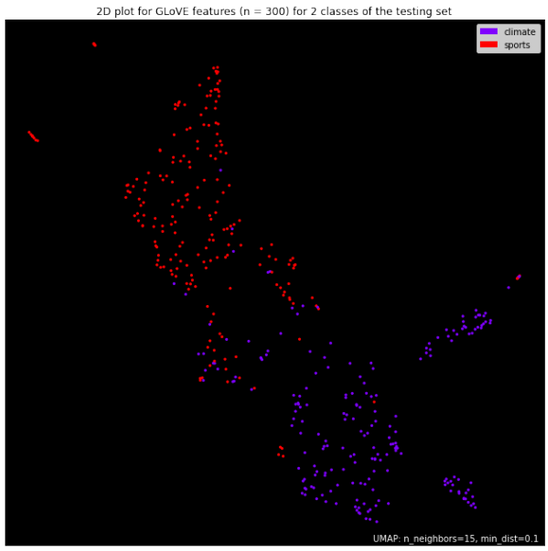

Figure 25: 2D Plot for GLoVE Testing set

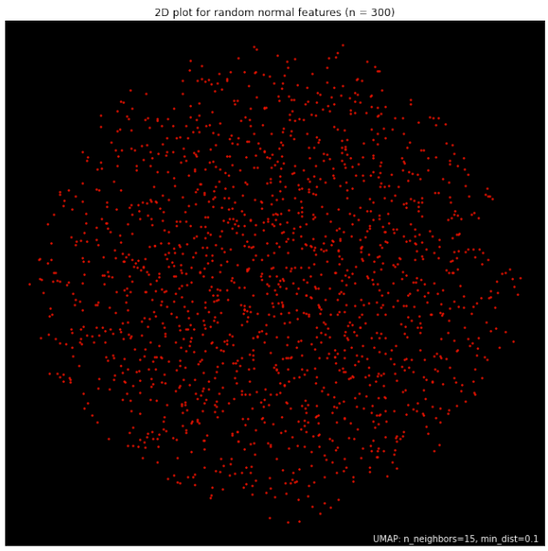

Figure 26: 2D Plot for GLoVE Testing set

We can observe clusters in both training and test set. This means that using GLoVE

embeddings are providing distinguishing features for classifiers to learn.
As expected, the randomly generated data are evenly distributed and we do not see any clustering being formed.