ECE219 Project 1

Jayanth **SHREEKUMAR**
UID: 805486993
Narayan **GOPINATHAN**
UID:605624014
Yuheng **HE**
UID: 505686149

January, 2022

Introduction

In this project, we design an end-to-end pipeline to classify news articles using different statistical classifiers. With the dataset, we explore the performance of these algorithms using both root labels, which is considered as a binary classification case, and leaf labels, which is taken as a multi-class classification procedure. The dataset provided is considered as a large-scale dataset and thus needs preprossessing and dimensionality reduction before applying the classifiers. In this project, we also evaluate how different feature extraction tools affect the outcome of the classification.

The project consists of several parts:

1. Dataset Split: We split the data into training set, which is used to train our model, and testing set, which is used to evaluate the performance of our model.
2. Feature Extraction: We Construct The Term Frequency-Inverse Document Frequency (TF-IDF) Model to extract the features of the textual data as an representation.
3. Dimensionality Reduction: We perform dimensionality reduction using different algorithms such as Latent Semantic Indexing (LSI) and Non-negative Matrix Factorization (NMF) and use the reduced data for following procedure.
4. Data Classification: We apply different classifier such as linear Support Vector Machine (SVM), Logistic Regression, and Naïve Bayes Model on binary classification case and multi-class SVM (with both One VS One and One VS the rese methods) and Naïve Bayes Model on multiclass classification case.
5. Grid Search: we tune the hyperparameters for our pipeline.
6. Pipeline Evaluation: We evaluate our model by printing out different metrics such as confusion matrix, F-1 score, accuracy score, recall score and precision score. These scores provide us a clear insight of how the model performs with different methods.
7. Word Embedding: We apply pre-training to a downstream classification task and evaluate comparative performance.

Getting Familiar with the Dataset

In this section, we evaluate the basics of the dataset such as its size and dimension, as well as the data distribution.

Question 1

- There are 2072 rows in the dataset and 9 columns, for a total of 18648 entries. Most of the entries in the database have fewer than 3000 words.
- See Fig. 1 and Fig. 2.
- The database was evenly divided into two roots and eight classes, with the exact same number for both roots and all eight classes. The number for root labels are 1036 and the one for leaf labels are 259.

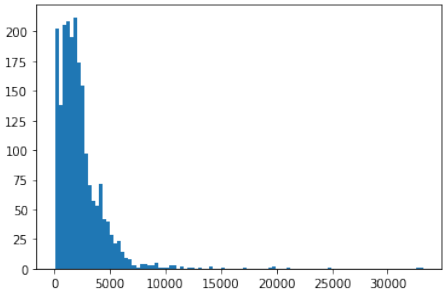
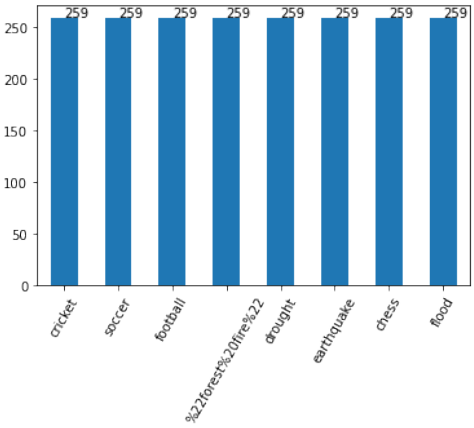
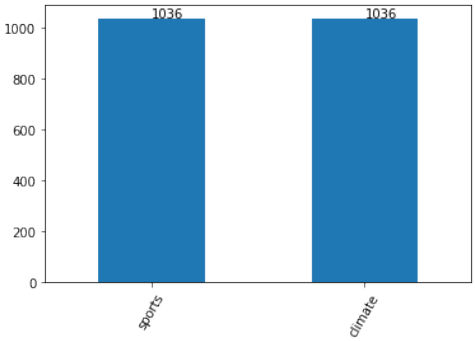


Figure 1: The total number of alphanumeric characters per row



(a) Number of entries for each class



(b) The total number of entries for each root

Figure 2: The number of entries in root and leaf classes

Binary Classification

Data Split

In this section, we randomly shuffle the data and then split the dataset into training set and testing set. The random shuffling is to make sure that coincidental pattern in the data is removed. Fig. 1 and Fig. 2 give us an insight of how the topics are distributed. From the plots we can see that they are balanced and evenly distributed.

Question 2

There are 1657 samples in the training set and 415 samples in the test set.

Feature Extraction

In this section, we transform our labels into binary classes, with Sports as class 0, and Climates as class 1.

An important procedure in classifying a corpus of texts is to extract the representing features of each data point. A good representation should remove redundancy of the raw features, and retain defining information for proper classification and to avoid overfitting.

Before performing the feature extraction process, we need to sanitize the text so that the useless information such as HTML artifacts, filler words and numeric terms are excluded. This helps us to downsize the textual entries and remove undesired redundancy, and it improves the feature extraction outcome.

We then perform lemmatization on the data, and after the lemmatization process, we vectorize our results and convert it into a matrix on which we can apply our classifiers. The parameter stopwords are set to 'English' so that words that are commonly used but contribute trivially to the textual features are removed.

Question 3

- Stemming is a simple algorithm that is prone to error. For example, in English, stemming would remove the suffix -ed from a verb in the past tense. However, this would lead to error in the case of irregular verbs like **went** as the past tense of **go**. Lemmatization would lead all verbs to be changed to their root form, so that **went** would become **go**. This is a much more complex operation and requires a bigger database of words in the language.
- min.df is the minimum document frequency for a word to be included in the analysis. Words that have a frequency below the threshold are ignored. For this reason, if you increase min.df then the matrix size decreases. Conversely, max.df is the maximum document frequency for a word to be included in the analysis, and so decreasing max.df decreases matrix size. Words that appear less frequently than the min.df or more frequently than the max.df in the document are excluded from the analysis.
- Stopwords, punctuations, and numbers should be removed before lemmatizing because that will reduce the amount of computational power required.
- The training matrix is 1657×10236. The testing matrix is 415×10236.

The resulting matrix size for the training dataset is 1657×10236 and the one for the testing dataset is 415×10236. The sizes are to big to apply any classifiers because it will be computationally expensive and algorithmically inefficient. Therefore, we need a dimensionality reduction step to retain the information of the matrices while removing redundancy.

Dimensionality Reduction

Classical classifiers such as Support Vector Machine, Logistic classifier, and Naïve Bayes Classifier may perform lousily with the high-dimensional matrix, and thus we need to project the data into lower dimensional space with the most relevant information. Additionally, we need to retain as much useful information as possible and in the meantime minimize the dimension. We test our dimensionality reduction functions with different k values, which is defined as the number of preserved components, to optimize the reduction process.

LSI: In this section, we use TruncatedSVD for the LSI dimensionality reduction method and preserve k components. However, since TruncatedSVD is preserving the components without centering, we need to center the data first so that the explained variance ratios in accordance to different k values are centered.

Moreover, since we only have 1657 samples, the maximum components preserved can not exceed this limit and thus we can only have 1657 preserved components. The result of the explained variance ratio is reported in Fig. 3.
NMF: In this section we apply NMF to reduce our dimensionality. This process finds the non-negative matrices that minimizes the Forbenius norm of the matrix.

Question 4

- The explained variance ratio is shown in Fig. 3. It it monotonically decreasing. The plot's concavity suggests that as k increases, the EVR drops drastically, but after a certain point it reaches an asymptote, as it approaches zero.
- The residual MSE error of NMF is larger, because LSI features Singular Value Decomposition (SVD) and it has been proven that the resulting matrix has the lowest error. NMF tries to approximate the data matrix and the process is random, it is not guaranteed that the error is the global minimum and therefore, NMF has a larger error than LSI.

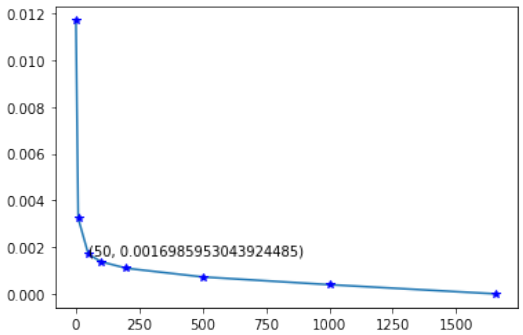


Figure 3: The Explained variance ratio with different k values

Classification Algorithms

Support Vector Machines

In this question, a hard-margin SVM, and a soft-margin SVM are trained with linear kernels. We perform 5-fold cross-validation to find the optimal gamma values for the SVMs, and finally compare the two models.

The SVM finds a linear decision boundary between the two classes such that the gap between the support vectors and the boundary is maximized. A hard-margin SVM with $\gamma >> 1$ severly penalizes a model for misclassification, while the soft-margin SVM allows for some misclassification as long as most of the data is classified correctly.

γ	1000	0.0001	100000	10 (optimal)
Accuracy	0.97590	0.49638	0.97349	0.98072
Precision	0.97597	0.24819	0.97363	0.98079
Recall	0.97586	0.5	0.97344	0.98068
F-1	97590	0.33172	0.973488	0.98072

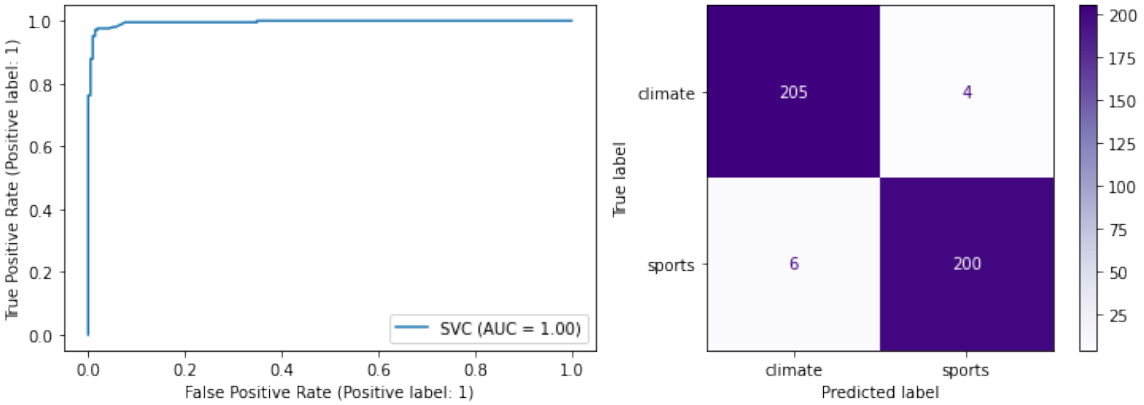
Table 1: Scores for different γ values

Question 5

- The ROC curves and confusion matrices correspond to $\gamma = 1000$, $\gamma = 100000$, $\gamma = .0001$ are shown in Fig. 4-6, respectively. And the scores of the classifiers correspond to $\gamma = 1000$, $\gamma = 100000$, $\gamma = .0001$ are given in Table 1.The hard-margin SVMs with high values of γ perform better. Of the three values we tested, the value $\gamma = 1000$ gives the highest values for accuracy, precision, recall,and F-1 score. The value $\gamma = 100000$ gives values that are almost equivalent but slightly less. All of these value are roughly equal to .975.

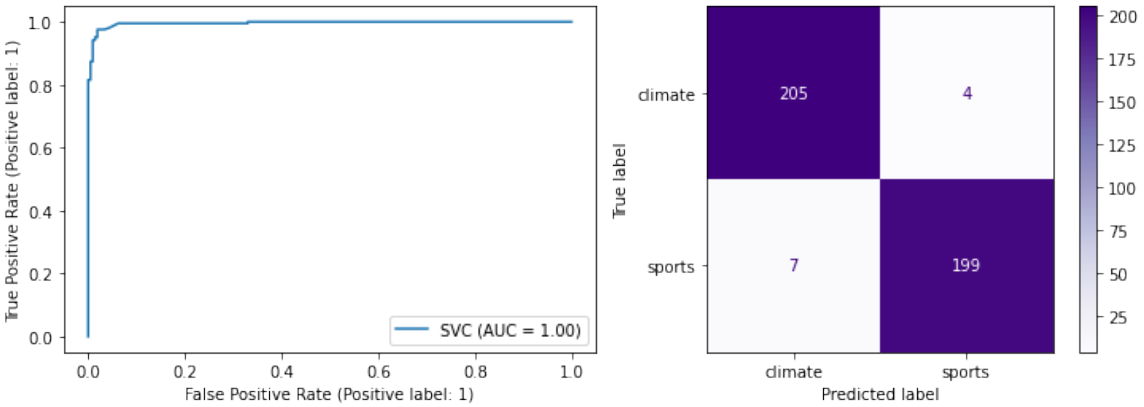
  On the other hand, the soft-margin SVM with a low value of γ perform much worse. When $\gamma = .0001$, the accuracy, precision, and recall, are all less than or equal to 0.5. In the confusion matrix, the algorithm incorrectly assumes that every article is sports. The soft margin has a less strict penalty on incorrectly classified elements. With soft margin $\gamma = .0001$, the classifier fails to represent the classes and thus results in underfitting. As such the soft margin classified is not competitive.

- From the process we discover that the best value for γ is 10. The ROC curve and the confusion matrix which correspond to this value are reported in Fig. 7, and the scores are reported in Table 1.



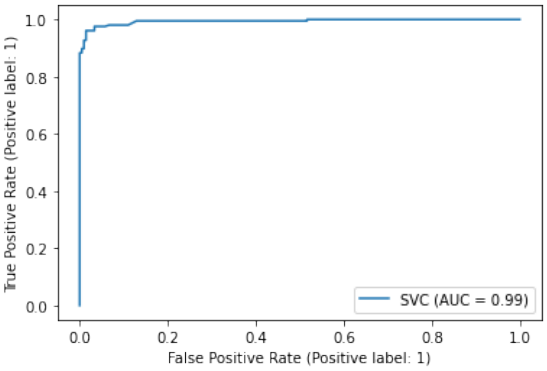
(a) ROC (b) Confusion matrix

Figure 4: ROC and Confusion matrix for $\gamma = 1000$

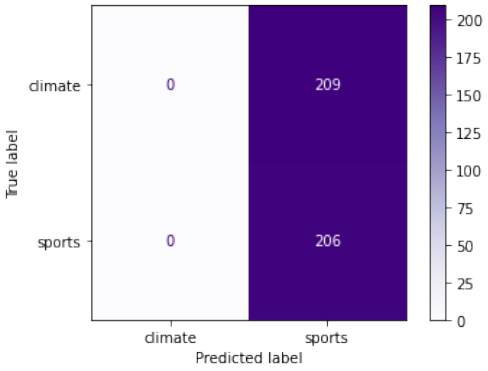


(a) ROC (b) Confusion matrix

Figure 5: ROC and Confusion matrix for $\gamma = 100000$

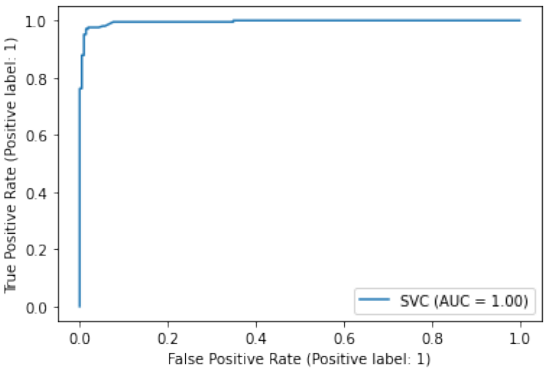


(a) ROC

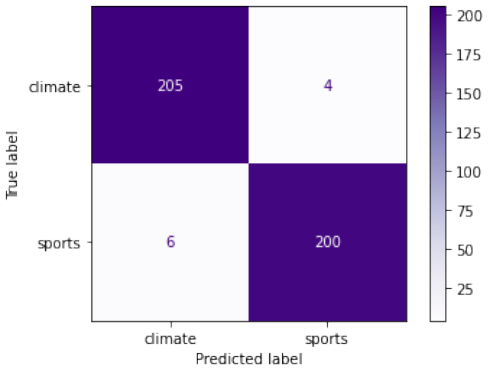


(b) Confusion matrix

Figure 6: ROC and Confusion matrix for $\gamma = .0001$



(a) ROC

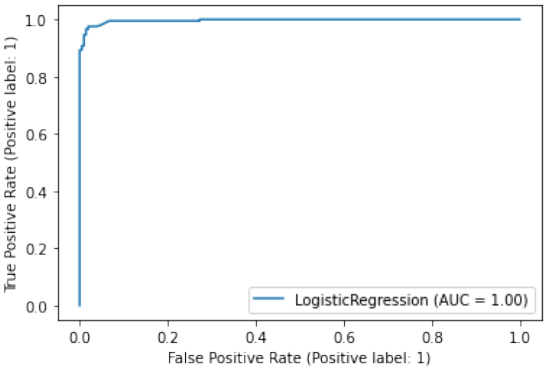


(b) Confusion matrix

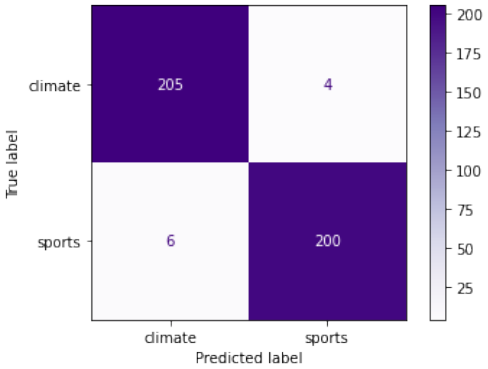
Figure 7: ROC and Confusion matrix for $\gamma = 10$

Logistic Regression

Logistic regression is a statistical model that uses a sigmoid function to obtain optimal parameters to maximize the likelihood of correct binary classification. A regularization term is used to penalize the model for using large weights and to prevent overfitting and facilitate generalization.



(a) ROC

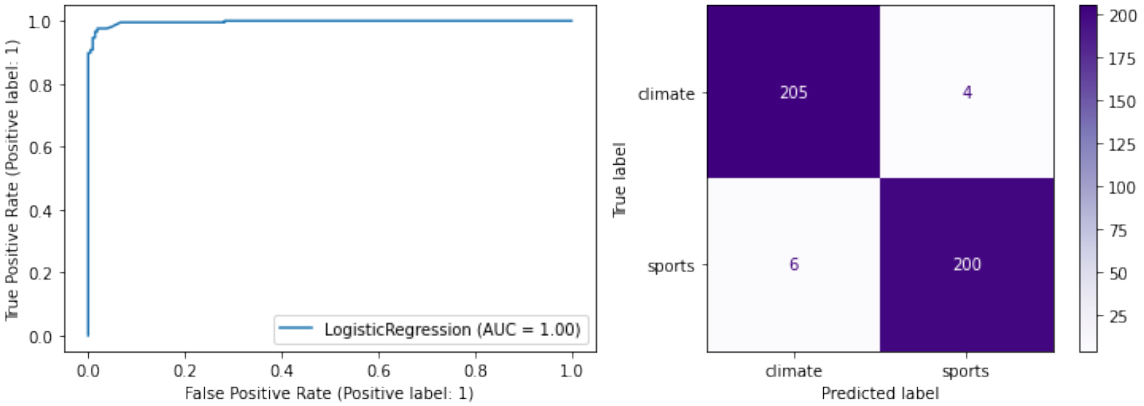


(b) Confusion matrix

Figure 8: ROC and Confusion matrix for logistic classifier without regularization

	No Regularization	L1 Regularization	L2 Regularization
λ	10^{62}	10 (optimal)	10 (optimal)
Accuracy	0.97590	0.97590	0.98072
Precision	0.97597	0.97597	0.98079
Recall	0.97586	0.97586	0.98068
F-1 Score	0.97590	0.97590	0.98072

Table 2: Scores for Logistic Regression with and without Regularization



(a) ROC

(b) Confusion matrix

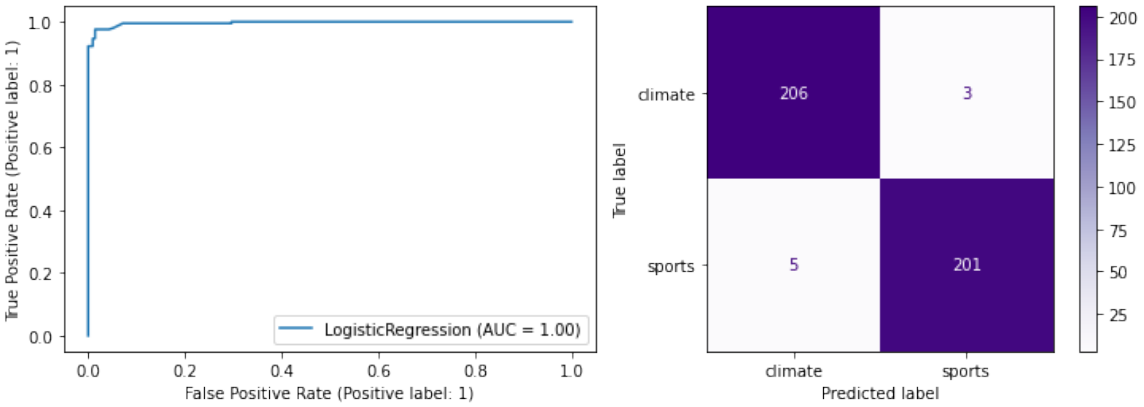
Figure 9: ROC and Confusion matrix for logistic classifier with L1 regularization

Question 6

- The ROC curves and confusion matrices correspond to logistic classifier without regularization, with L1 regularization, and with L2 regularization are shown in Fig. 8-10, respectively. The scores are reported in Table 2.
- Both L1 regularization and L2 regularization found that the best value for γ is 10.
- L2 regularization has the highest accuracy, precision, recall, and F1 Score of all three logistic classifiers. All three classifiers have values for all four parameters above .97, but the L2 regularization is the only one that has a value above .98.
- Regularization is a means to penalize the model for utilizing heavy weights. As a model that has regularization is being penalized for misclassification, it is forced to make a simple "fit" to the data. This means that, when implemented correctly, regularization should not have a large impact on test accuracy. Using a larger regularization parameter means that the learnt coefficients are smaller compared to non-regularized coefficients. The main differences between L1 and L2 regularization are as follows :

  L1-Regularization helps in feature selection by eliminating features that are not important. Also, it is suitable for models to train on sparse matrices that are often seen in NLP applications. It is also more robust to outliers that L2 regularization.

  L2-Regularization leads to denser models compared to when using L1 regularization. The main advantage of L2 regularization is that it has a closed form solution due to the squaring of terms and is less computationally expensive. However, the main drawback is that it is not robust to outliers due to the squaring of values.



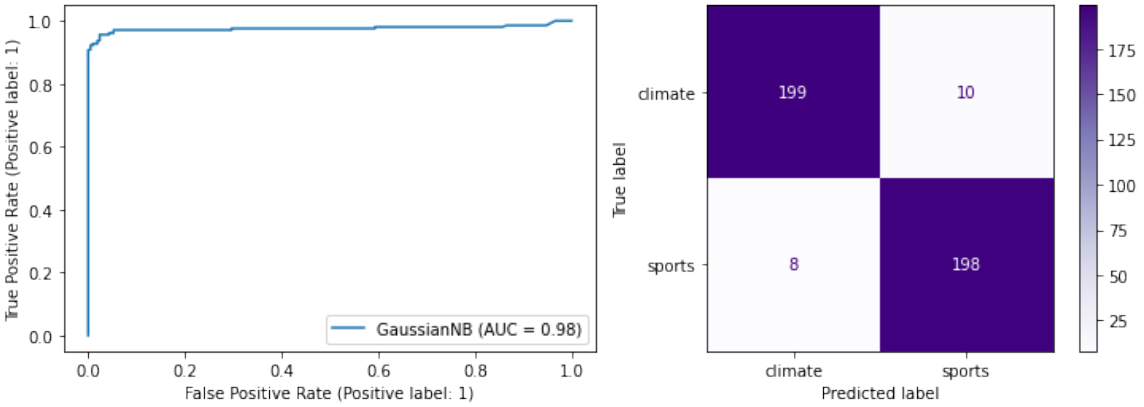
(a) ROC (b) Confusion matrix

Figure 10: ROC and Confusion matrix for logistic classifier with L2 regularization

Naïve Bayes Model

Question 7

- The ROC curve and the confusion matrix are reported in Fig. 11, and the scores are given in Table 3. We can see that Naïve Bayes classifier can successfully classify the cases with mild misclassification, the metrics that can be used to evaluate the overall performance of the classifier are reported in Table 8. As is shown, the accuracy rate is approximately 95.66%, which implies that the classifier is performing well.



(a) ROC (b) Confusion matrix

Figure 11: ROC and Confusion matrix for logistic classifier with Naïve Bayes classifier

Naïve Bayes Classifier	
Accuracy	0.956627
Precision	0.956638
Recall	0.956659
F-1 Score	0.956626

Table 3: Scores for Naïve Bayes Classifier

Grid Search of Parameters

Grid search is a useful technique to find out the optimal combination of hyper parameters across different modules of a pipeline simultaneously.

Question 8

- Our pipeline consisted of the following modules:

```
pipe = Pipeline([('vectorizers', vectorizer),('tfid', TfidfTransformer()),\
        ('dim_reds', lsi_5),('classifiers',gaussian_naive_bayes)], memory=memory)
```

  The total number of combinations included in grid search was 144 each(once on cleaned data and once again on raw data) for a total of 288 combinations.

- Grid search was performed with the parameter cv set to 5 to perform 5-fold cross validation using the following line of code:

```
grid = GridSearchCV(pipe, cv=5, n_jobs=1, param_grid=parameters,\
                            scoring='accuracy')
```

- The full list of modules and their parameters are described below.
- According to our search, the 5 best models in terms of highest validation accuracy for cleaned data are in Table 4 and the 5 best models in terms of highest validation accuracy for raw dat are in table 5.

Rank	min df	Feat. Extract	Dimensionality Reduction	Classifier	Val. Acc.	Test Acc.
1	3	Stemmer	TruncatedSVD (n components = 500, n iter = 20, random state = 42)	LogisticRegression(C=10, penalty='l1', solver='liblinear')	0.9728	0.9735
2	3	Nothing	TruncatedSVD (n components = 500, n iter=20, random state = 42)	LogisticRegression(C=10, penalty='l2', solver='liblinear')	0.9698	0.9807
3	5	Lemmatize	TruncatedSVD (n components = 500, n iter = 20, random state = 42)	LogisticRegression(C=10, penalty='l2', solver='liblinear')	0.9698	0.9759
4	5	Nothing	TruncatedSVD (n components = 500, n iter = 20, random state = 42)	SVC(C=10, kernel='linear')	0.9692	0.9759
5	5	Nothing	TruncatedSVD (n components=500, n iter=20, random state=42)	LogisticRegression(C=10, penalty='l2', solver='liblinear')	0.9692	0.9807

Table 4: Best Performed Hyper-parameters in terms of Validation Accuracy on Cleaned Data and their testing accuracies. Refer to the Colab Notebook for Precision, Recall, F1-Score, and Confusion Matrix and ROC Curves.

The modules and their hyper-parameters are:

1. We called the whole pipeline twice, once on cleaned data, and once on raw data without passing it through the regex cleaning process.

2. Vectorizers: The value of min_df for CountVectorizer() was either 3 or 5.

3. Feature Extractors: We designed Lemmatizer() and Stemmer() as classes that could be used by CountVectorizer(). To implement "No Compression", we simply called CountVectorizer() without any argument for the parameter tokenizer().Here are the defined variations:

```
stem_vectorizer_3 = CountVectorizer(tokenizer=Stemmer(),
                          stop_words = 'english',
                          lowercase = True, min_df = 3)

stem_vectorizer_5 = CountVectorizer(tokenizer=Stemmer(),
                          stop_words = 'english',
                          lowercase = True, min_df = 5)

lemma_vectorizer_3 = CountVectorizer(tokenizer=LemmaTokenizer(),
                          stop_words = 'english',
                          lowercase = True, min_df = 3)

lemma_vectorizer_5 = CountVectorizer(tokenizer=LemmaTokenizer(),
                          stop_words = 'english',
                          lowercase = True, min_df = 5)

vectorizer_3 = CountVectorizer(stop_words = 'english',
                          lowercase = True, min_df = 3)

vectorizer_5 = CountVectorizer(stop_words = 'english',
                          lowercase = True, min_df = 5)
```

Rank	min df	Feat. Extract	Dim. Reduction	Classifier	Val. Acc.	Test Acc.
1	3	Stemmer	TruncatedSVD (n components = 500, n iter = 20, random state = 42)	LogisticRegression(C=10, penalty='l1', solver='liblinear')	0.9728	0.9663
2	3	Nothing	TruncatedSVD (n components = 500, n iter=20, random state = 42)	LogisticRegression(C=10, penalty='l2', solver='liblinear')	0.9698	0.9807
3	5	Lemmatize	TruncatedSVD (n components = 500, n iter = 20, random state = 42)	LogisticRegression(C=10, penalty='l2', solver='liblinear')	0.9698	0.9759
4	5	Nothing	TruncatedSVD (n components = 500, n iter = 20, random state = 42)	SVC(C=10, kernel='linear')	0.9692	0.9783
5	5	Nothing	TruncatedSVD (n components=500, n iter=20, random state=42)	LogisticRegression(C=10, penalty='l2', solver='liblinear')	0.9692	0.9831

Table 5: Best Performed Hyper-parameters in terms of Validation Accuracy on Raw Data and their testing accuracies. Refer to the Colab Notebook for Precision, Recall, F1-Score, and Confusion Matrix and ROC Curves.

4. Dimensionality reduction: We considered all combinations of Truncated-SVD and NMF with n-componenets [5, 50, 500]. Here are the defined variations:

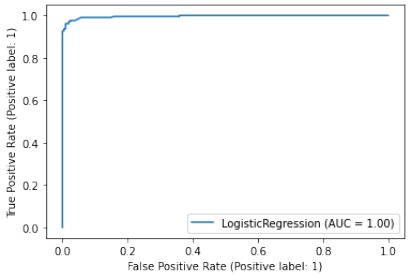
```
lsi_5 = TruncatedSVD(n_components=5, n_iter=20, random_state=42)
lsi_50 = TruncatedSVD(n_components=50, n_iter=20, random_state=42)
lsi_500 = TruncatedSVD(n_components=500, n_iter=20, random_state=42)

nmf_5 = NMF(n_components=5, init='random', random_state=42)
nmf_50 = NMF(n_components=50, init='random', random_state=42)
nmf_500 = NMF(n_components=500, init='random', random_state=42)
```

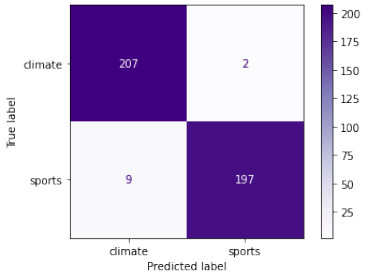
5. Classifiers: We experimented with the following 4 classifiers:

```
svm_model = svm.SVC(kernel = "linear", C=best_gamma)
logistic_regression_l1 = LogisticRegression(penalty='l1', \
            solver = 'liblinear', C = best_lambda_l1)
logistic_regression_l2 = LogisticRegression(penalty='l2', \
            solver = 'liblinear', C = best_lambda_l2)
gaussian_naive_bayes = GaussianNB()
```

The confusion matrices and ROCs for the 5 best models on **Cleaned Data**:

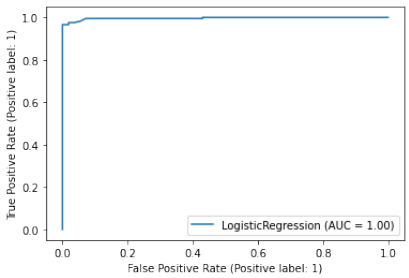


(a) ROC

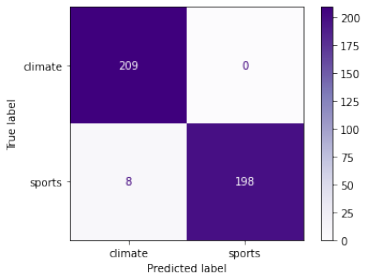


(b) Confusion matrix

Figure 12: ROC and Confusion matrix for Best Combination on Cleaned Data

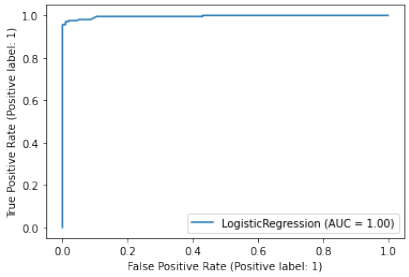


(a) ROC

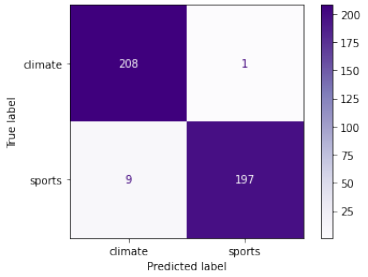


(b) Confusion matrix

Figure 13: ROC and Confusion matrix for 2nd Best Combination on Cleaned Data

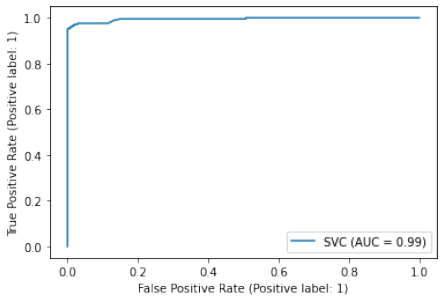


(a) ROC

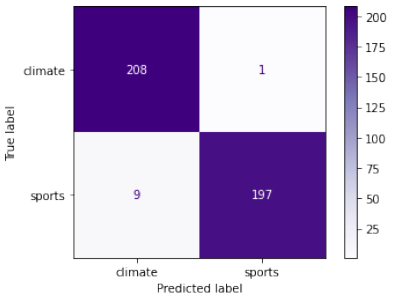


(b) Confusion matrix

Figure 14: ROC and Confusion matrix for 3rd Best Combination on Cleaned Data

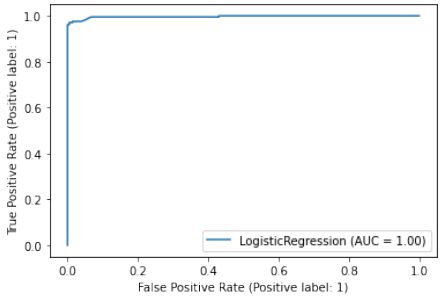


(a) ROC

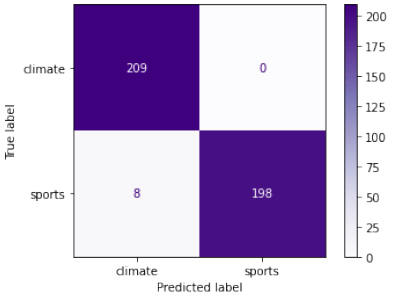


(b) Confusion matrix

Figure 15: ROC and Confusion matrix for 4th Best Combination on Cleaned Data



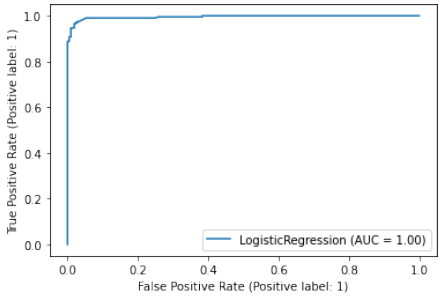
(a) ROC



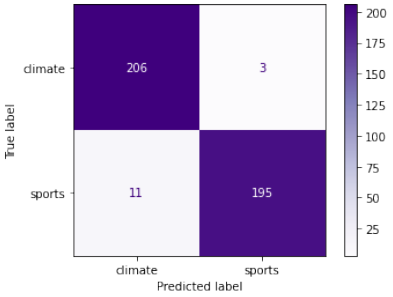
(b) Confusion matrix

Figure 16: ROC and Confusion matrix for 5th Best Combination on Cleaned Data

The confusion matrices and ROCs for the 5 best models on **Raw Data**:

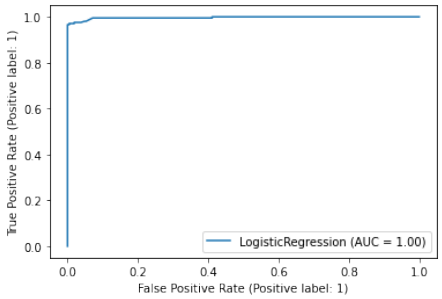


(a) ROC

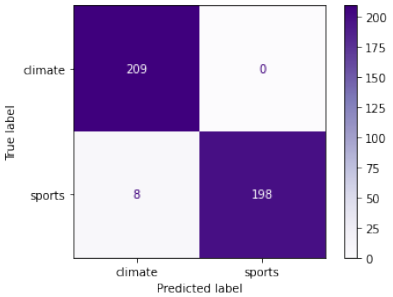


(b) Confusion matrix

Figure 17: ROC and Confusion matrix for Best Combination on Raw Data

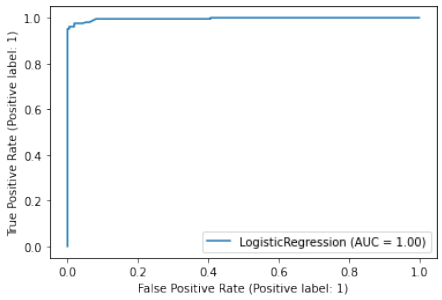


(a) ROC

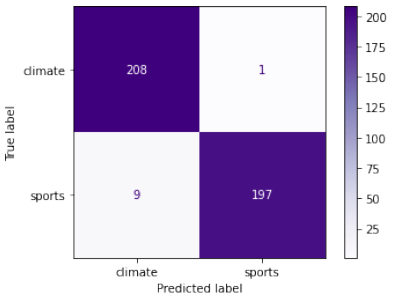


(b) Confusion matrix

Figure 18: ROC and Confusion matrix for 2nd Best Combination on Raw Data

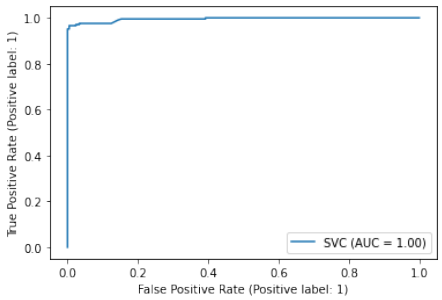


(a) ROC

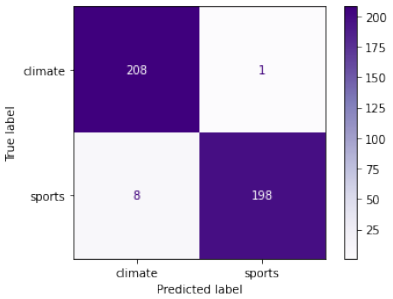


(b) Confusion matrix

Figure 19: ROC and Confusion matrix for 3rd Best Combination on Raw Data

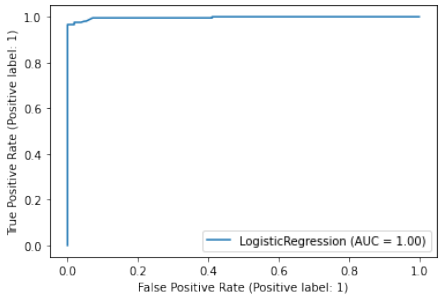


(a) ROC

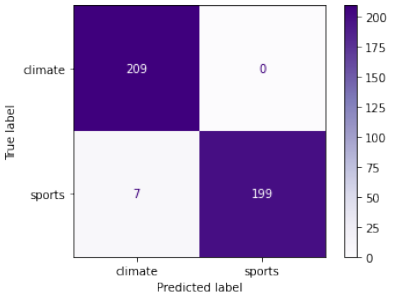


(b) Confusion matrix

Figure 20: ROC and Confusion matrix for 4th Best Combination on Raw Data



(a) ROC



(b) Confusion matrix

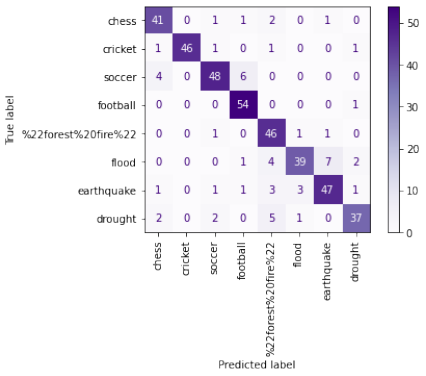
Figure 21: ROC and Confusion matrix for 5th Best Combination on Raw Data

Multiclass Classification

In this section we perform multiclass classification methods such as Naïve Bayes classification and multi-class SVM classification with both One VS One and One VS the Rest

Question 9

- The confusion matrix and the accuracy, recall, precision and F-1 score of the classifiers are shown in Fig. 22-27, respectively.
  Among the three classifiers, SVM One VS the Rest has the best performance (in Figure 26), meaning that it obtains the highest accuracy rate. Such a result makes sense in that this method allows SVM to compare the label with the rest of the classes and such a process gives the classifier a better understanding of the general dataset, leading to a higher accuracy rate.
  The class imbalance issue in SVM One VS the Rest is handled by the built-in library so we do not see any imbalance issue raised by using this method.
- We produce the confusion matrices in the form of 8 by 8 matrices and
  - The major diagonal entries have the greatest values. This makes sense because it means most of the cases are classified correctly. However, there are some visible blocks on the major diagonal. For example, in the multiclass Naïve Bayes Classifier, we notice that a block exists among the classes flood and earthquake. This is because many texts include information about floods and earthquakes, so the classifier confused those categories.
  - With the previous observation, we can merge these two classes together and perform Naïve Bayes Classification to see if it can produce a better accuracy rate. The confusion matrices and the scores are displayed in Fig. 28-31, respectively.
  - The class imbalance issue in this case has not affected the performance of the classifier. On the contrary, after merging the labels into a new label class, the performance of the classifier has improved, which means the two merged labels do have a latent correlation.

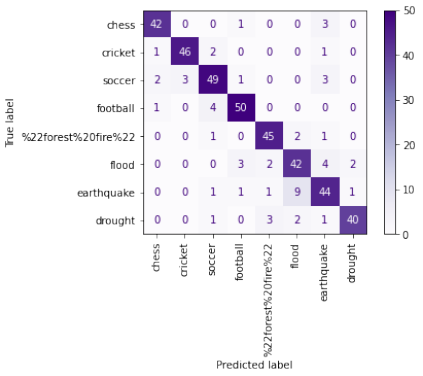


(a) Naive Bayes LSI Confusion matrix

Naïve Bayes Classifier with LSI	
Accuracy	0.826265
Precision	0.867933
Recall	0.863391
F-1 Score	0.862298

(b) Scores for Naive Bayes LSI

Figure 22: Naive Bayes Classifier with LSI

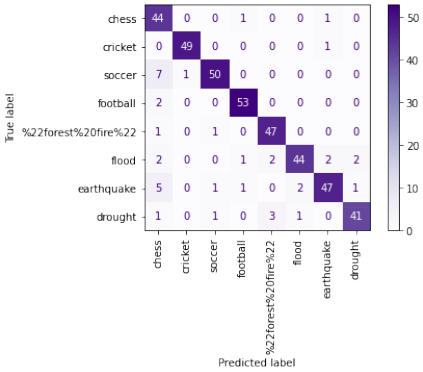


(a) Naive Bayes NMF Confusion matrix

Naïve Bayes Classifier with NMF	
Accuracy	0.862651
Precision	0.867207
Recall	0.86507
F-1 Score	0.865833

(b) Scores for Naive Bayes NMF

Figure 23: Naive Bayes Classifier with NMF

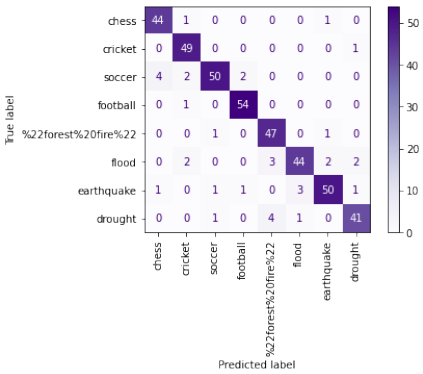


(a) SVM One vs One with LSI Confusion matrix

SVM One v One with LSI	
Accuracy	0.903614
Precision	0.909113
Recall	0.906063
F-1 Score	0.904104

(b) Scores for SVM One vs One with LSI

Figure 24: SVM One vs One with LSI

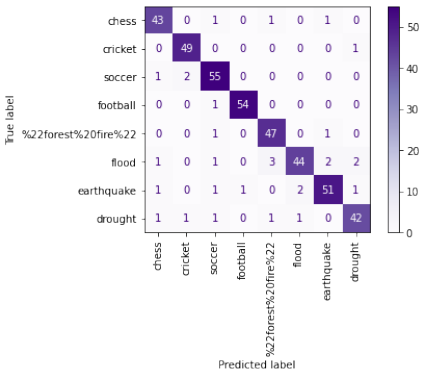


(a) SVM NMF Confusion matrix

SVM One v One with NMF	
Accuracy	0.884337
Precision	0.893314
Recall	0.88626
F-1 Score	0.884654

(b) Scores for SVM NMF

Figure 25: SVM One vs One Classifier with NMF

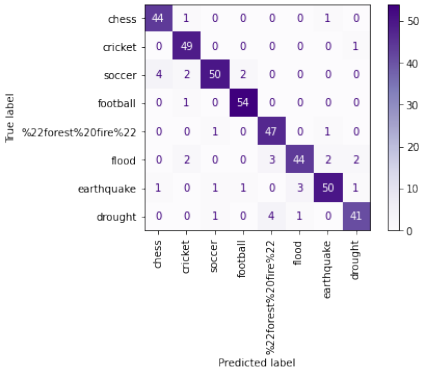


(a) SVM One v Rest with LSI Confusion matrix

SVM One v Rest with LSI	
Accuracy	0.927711
Precision	0.927624
Recall	0.927825
F-1 Score	0.927042

(b) Scores for SVM One v Rest LSI

Figure 26: SVM One vs Rest Classifier with LSI

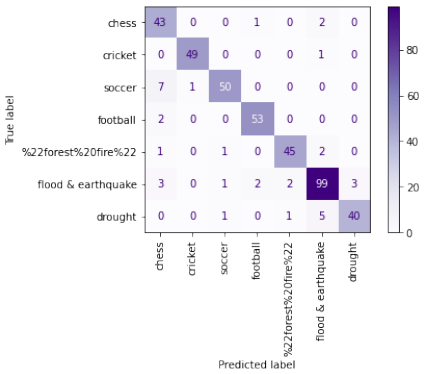


(a) SVM One v Rest NMF Confusion matrix

SVM One v Rest with NMF	
Accuracy	0.913253
Precision	0.912963
Recall	0.914914
F-1 Score	0.912619

(b) Scores for One v Rest SVM NMF

Figure 27: SVM One v Rest Classifier with NMF

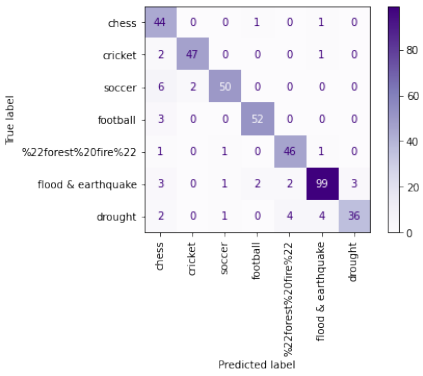


(a) Merged 1v1 LSI Confusion matrix

Merged Case One v One with LSI	
Accuracy	0.913253
Precision	0.916239
Recall	0.915703
F-1 Score	0.914261

(b) Scores for Merged 1v1 with LSI

Figure 28: Confusion matrix and scores for merged 1 v 1 classifier with LSI

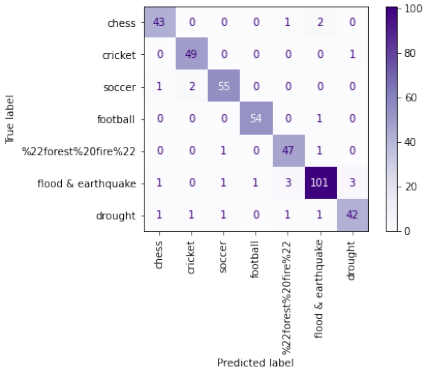


(a) Merged 1 v 1 NMF Confusion matrix

Merged Case One v One with NMF	
Accuracy	0.901205
Precision	0.901571
Recall	0.901254
F-1 Score	0.897578

(b) Scores for Merged 1v1 NMF

Figure 29: Confusion matrix and scores for merged 1 v 1 Classifier with NMF

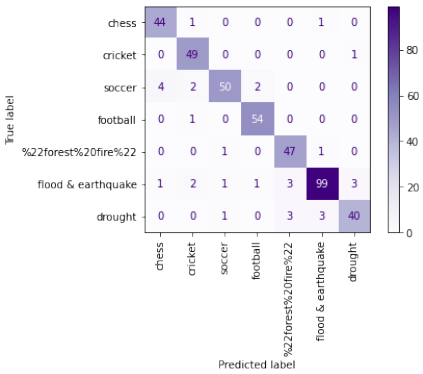


(a) Merged 1 v R LSI Confusion matrix

Merged Case One v Rest with LSI	
Accuracy	0.942169
Precision	0.939558
Recall	0.945123
F-1 Score	0.942109

(b) Scores for Merged 1vR LSI

Figure 30: Confusion matrix and scores for merged 1 v R Classifier with LSI



(a) Merged 1 v 1 NMF Confusion matrix

Merged Case One v Rest with NMF	
Accuracy	0.922891
Precision	0.918206
Recall	0.927367
F-1 Score	0.921537

(b) Scores for Merged 1vR NMF

Figure 31: Confusion matrix and scores for merged 1 v R Classifier with NMF

Word Embedding

GLoVE stands for Global Vectors for Word Representation and is an unsupervised learning algorithm that utilizes global context as well as local statistics to create vector representations for words. The main intuition is that ratios of word co-occurences also encode meaning. Semantically similar words have smaller distance between them, while semantically different words are separated in space.

In this section, we utilize pretrained GLoVE embeddings to describe each document as well as train and evaluate classifier models on these GLoVE embeddings to distinguish between two classes: "Sports" and "Climate". We visualize the relationship between the dimension of GLoVE embedding used and the accuracy of a classifier model trained on these embeddings. Finally, we visualize the dataset in terms of its GLoVE embeddings and observe that semantically similar documents tend to form clusters.

Question 10

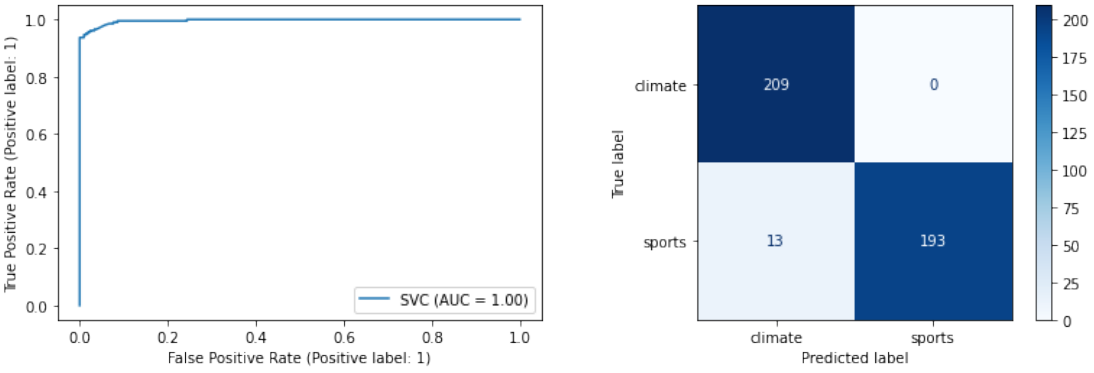
- GLoVE embeddings are trained on the ratio of co-occurence probabilities rather than the probabilities themselves as the ratios are better able to distinguish amongst contextual words that do and do not pertain to the word being encoded.
- No, the GLoVE embedding is not the same for the word "running" in the two cases as GLoVE captures global context along with local statistics. Thus, context words like "presidency" and "park" will distinguish the two GLoVE Embeddings for the word "running" based on context.
- We expect the values of $||queen - king||_2$ and $||wife - husband||_2$ to be very similar (ideally equal) as they are pairs of contextually similar words.
  Let $X = ||queen - king||_2$ and $Y = ||wife - husband||_2$
  Then $||queen - king - wife + husband||_2 = \sqrt{X^2 + Y^2 - 2XY}$. Values obtained: $||queen - king||_2 = 5.966258$ $||wife - husband||_2 = 3.1520464$ $||queen - king - wife + husband||_2 = 6.1650367$. Ideally, this should be 7.91 but it appears(as expected) that GLoVE embeddings are not completely ideal and so, we see the difference in values.
- Stemming is a very crude process that chops off the end of words and is not suited for GLoVE. Lemmatization does a much better job as it uses a dictionary to convert every word to its base form. However, lemmatization is not the right solution for this. The purpose of GloVe is to assess the relationship between related words, such as 'big', 'bigger', and 'biggest'. If we lemmatize, then those all become 'big'. As such, it is best to do nothing at all.

Question 11

- (a) The feature engineering pipeline that we use is as follows:
  1. Read and store pre-trained GLoVE embeddings.
  2. Create a function called "create_glove_features()" that creates a single GLoVE feature vector per document.
  3. Iterate over each document. Note that the dataset we use has been cleaned and lemmatized.
  4. For each document, iterate over the words in the document.
  5. For each word, look for the corresponding glove embedding in the stored GLoVE embeddings. If found, append it to a list called sentence_vecs.
  6. After iterating through all words in the document, take the mean of all the appended word embeddings to get an aggregate word embedding for that document.
  7. Append this aggregate word embedding to a list called glove_features.
  8. All of this is done on both train and test datasets.
  9. Finally use these embeddings to train and evaluate a classifier model.
- (b) We trained and tested 5 different classifier models, the resulting test accuracies of which of which are as follows:

	SVM	Logistic No Reg.	L1 Logistic	L2 Logistic	Naïve Bayes
Accuracy	0.96867	0.95421	0.96144	0.96626	0.93253
Precision	0.97072	0.95619	0.96376	0.96801	0.93413
Recall	0.96844	0.95398	0.96119	0.96605	0.93231
F-1 Score	0.96862	0.95414	0.96138	0.96622	0.93244

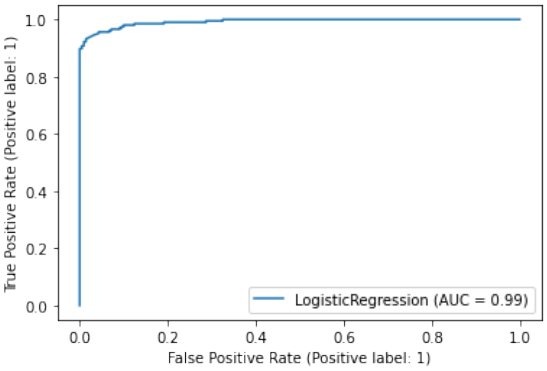
Table 6: Classifier Accuracies on GLoVE Features



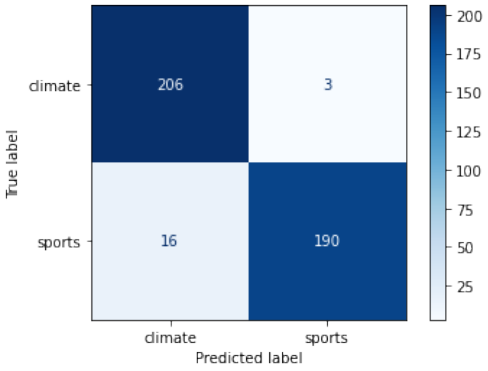
(a) ROC

(b) Confusion matrix

Figure 32: ROC and Confusion matrix for SVM classifier for GloVe features

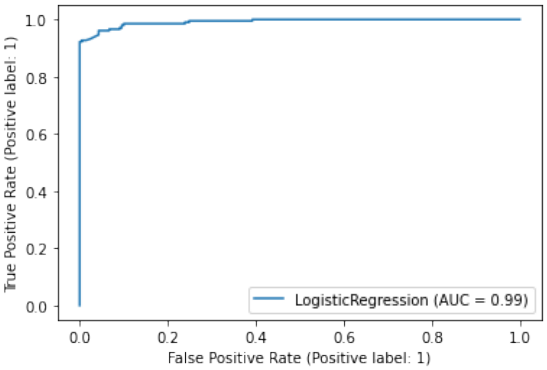


(a) ROC

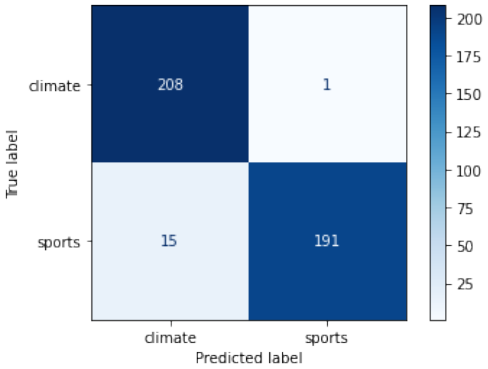


(b) Confusion matrix

Figure 33: ROC and Confusion matrix for Logistic Regression classifier without regularization for GloVe features

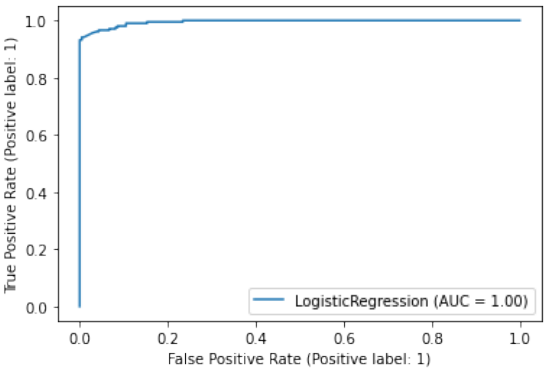


(a) ROC

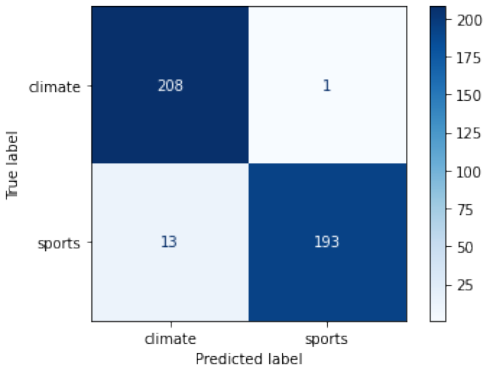


(b) Confusion matrix

Figure 34: ROC and Confusion matrix for Logistic Regression classifier with L1 regularization for GloVe features

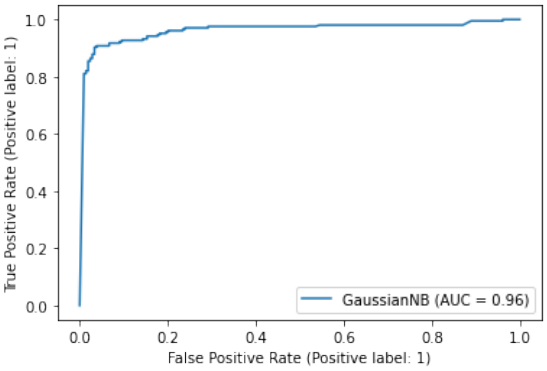


(a) ROC

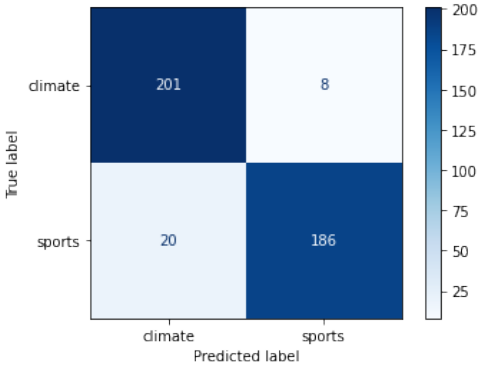


(b) Confusion matrix

Figure 35: ROC and Confusion matrix for Logistic Regression classifier with L2 regularization for GloVe features



(a) ROC



(b) Confusion matrix

Figure 36: ROC and Confusion matrix for Naive Bayes classifier for GloVe features

Question 12 H

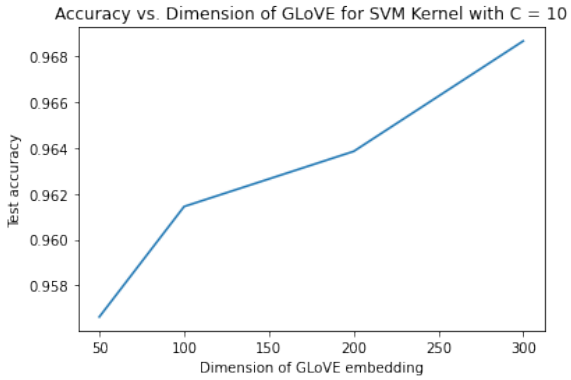
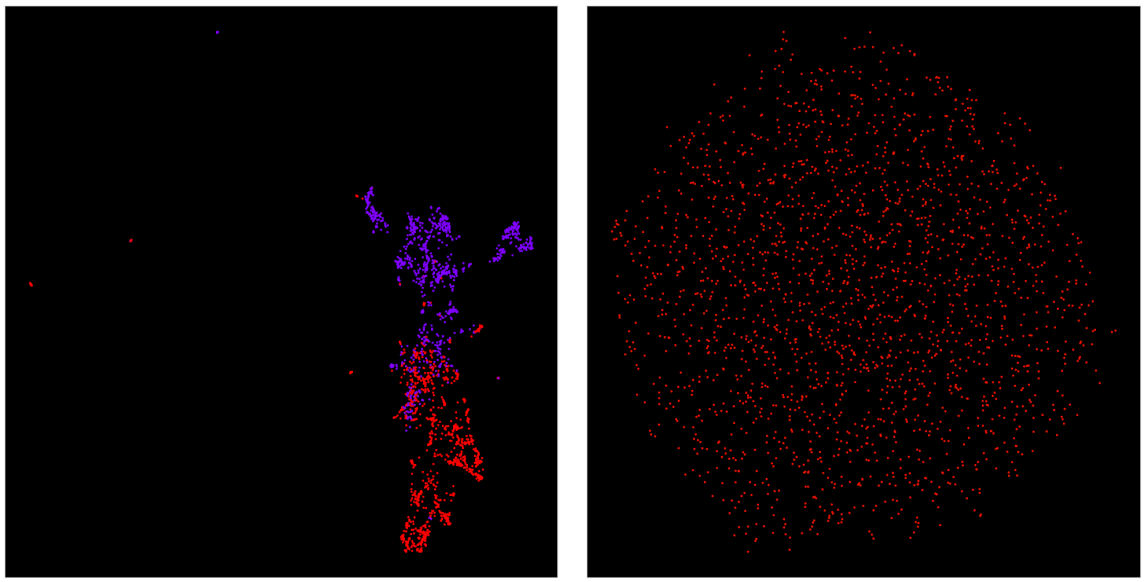


Figure 37: Dimension of GLoVE embeddings vs Test Accuracy

For this question, we used an SVM kernel with optimized gamma value. From the figure, we see that the test accuracy increases as the dimension of the GLoVE embedding increases. This is expected as larger dimensional embeddings are capable of capturing much more semantic information, which allows the classifier model to perform classification with more "knowledge".

Question 13



(a) GLoVE embedding vectors

(b) Random vectors

Figure 38: GloVe embeddings and normalized random vectors

From the figure on the left(GLoVE embeddings), we clearly see two separate clusters being formed. As these clusters are so distinct, this means that the classifier model can perform its task effectively. However, as expected, we do not see clusters in the randomly generated embeddings.